Type Safety for Solidity

Billy Thornton Diego Marmsoler

November 16, 2026
Department of Computer Science, University of Exeter, Exeter, UK
{bt319, d.marmsoler}@exeter.ac.uk

Abstract

This development studies type consistency and its preservation for the Isabelle/HOL semantics of Solidity. It defines consistency properties for values, memory, storage, environments, and states, and establishes supporting results for initialisation and copying. Within the stated locale assumptions, the statement preservation theorem shows that successful execution from a type-consistent, fully initialised environment satisfies the state invariant. Concrete examples illustrate the assumptions and the use of the theorem. The underlying Solidity semantics is imported as a separate session.
Keywords: Type safety, Smart contracts, Solidity, Program verification, Isabelle/Solidity

Contents

1 Introduction 3

2 Iteration lemmas (TypeSafe_Support) 5

3 Locations and sublocations (TypeSafe_Hashing_Subs) 7

4 Type consistency for value types (TypeSafe_Base_Types) 15

5 Type consistency for memory (TypeSafe_Memory) 21

6 Type consistency for storage (TypeSafe_Storage) 60

7 Type consistency for environments and states (TypeSafe_Def) 86

8 Storage-to-storage copies (TypeSafe_Copies) 100

9 Memory initialisation (TypeSafe_Memory_Init) 114

10 Compatibility of storage and memory types (TypeSafe_Mem_Sto_Comp) 144

11 Copies between memory and storage (TypeSafe_Memory_Copies) 145

12 Environment initialisation and declarations (TypeSafe_Environment) 220

13 Declarations (TypeSafe_Decl) 233

14 Expression preservation (TypeSafe_Expressions) 341

15 Supporting lemmas for statement preservation (TypeSafe_Statement_Support) 391

16 Preservation for assignments (TypeSafe_Assign) 409

17 Statement preservation (TypeSafe) 598

18 Examples (TypeSafe_Examples) 664

1 Introduction

This development builds on the Isabelle/HOL formalisation of Solidity by Marmsoler and Brucker [1]. The formalization is described in two accompanying papers [2, 3]. The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle. The structure follows the theory dependencies (see Figure 1).

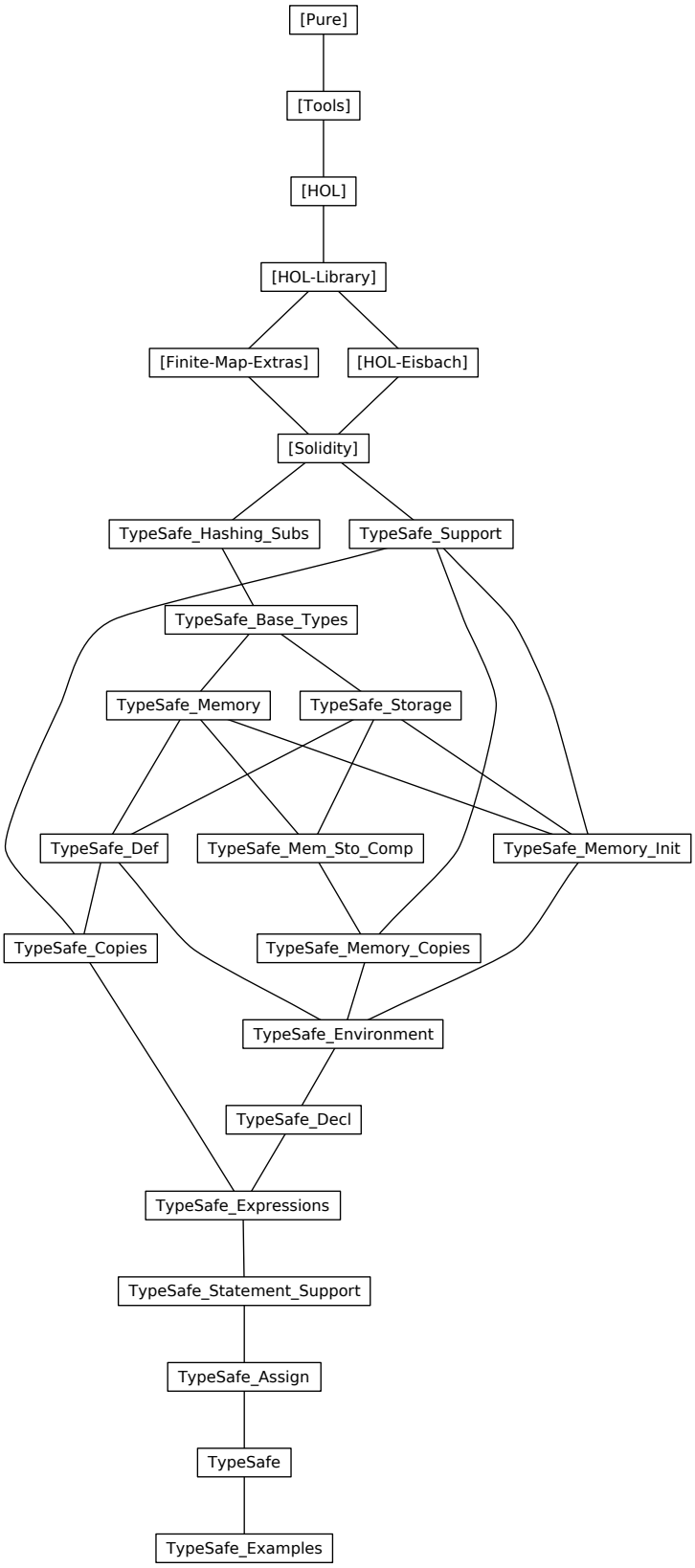


Figure 1: The Dependency Graph of the Isabelle Theories.

2 Iteration lemmas (TypeSafe_Support)

theory *TypeSafe_Support*
  imports *"Solidity.Solidity_Main"*
begin

Induction and invariant principles for *iter'* and *iter*, which drive the array-copying and initialisation functions of the semantics.

lemma *iter'_induct:*
  assumes *"⋀v'. iter' f v 0 = Some v' ⟹ P 0 v'"*
    "⋀x v''. (∃v'. iter' f v x = Some v' ∧ P x v' ∧ f x v' = Some v'') ⟹ iter' f v (Suc x) = Some v'' ⟹ P (Suc x) v''"
  shows *"iter' f v x = Some v' ⟹ P x v'"*
proof *(induction x arbitrary: v')*
  case *0*
  then show *?case* **using** *assms(1)* **by** *blast*
next
  case *(Suc x)*
  from *Suc.prems* **obtain** *v''* **where** *"f (Suc x - 1) v'' = Some v'"* **and** *"iter' f v x = Some v''"*
    using *iter'.simps[of f v "Suc x"]* **by** *(auto split: option.split_asm)*
  then have *"f x v'' = Some v'"* **and** *"iter' f v x = Some v''"* **by** *simp+*
  then show *?case* **using** *assms(2)[OF _ Suc.prems] Suc.IH* **by** *simp*
qed

lemma *iter'_invariant:*
  assumes *run: "iter' f v x = Some v'"*
    and *init: "I v"*
    and *step: "⋀i v1 v2. I v1 ⟹ f i v1 = Some v2 ⟹ I v2"*
  shows *"I v'"*
proof -
  have *"⋀v''. iter' f v x = Some v'' ⟹ I v''"*
  proof *(rule iter'_induct[***where** *P = "λ_ v''. I v''"])*
    fix *v''*
    assume *"iter' f v 0 = Some v''"*
    then have *"v'' = v"* **by** *simp*
    with *init* **show** *"I v''"* **by** *simp*
  next
    fix *i v''*
    assume *"∃v'. iter' f v i = Some v' ∧ I v' ∧ f i v' = Some v''"*
    then obtain *v1* **where** *"I v1"* **and** *"f i v1 = Some v''"* **by** *blast*
    then show *"iter' f v (Suc i) = Some v'' ⟹ I v''"* **using** *step* **by** *blast*
  qed
  with *run* **show** *?thesis* **by** *simp*
qed

lemma *iter'_indexed_invariant:*
  assumes *run: "iter' f v x = Some v'"*
    and *init: "P 0 v"*
    and *step: "⋀i v1 v2. P i v1 ⟹ f i v1 = Some v2 ⟹ P (Suc i) v2"*
  shows *"P x v'"*
proof -
  have *"⋀v''. iter' f v x = Some v'' ⟹ P x v''"*
  proof *(rule iter'_induct[***where** *P = P])*
    fix *v''*
    assume *"iter' f v 0 = Some v''"*
    then have *"v'' = v"* **by** *simp*
    with *init* **show** *"P 0 v''"* **by** *simp*
  next
    fix *i v''*
    assume *"∃v'. iter' f v i = Some v' ∧ P i v' ∧ f i v' = Some v''"*
    then obtain *v1* **where** *"P i v1"* **and** *"f i v1 = Some v''"* **by** *blast*
    then show *"iter' f v (Suc i) = Some v'' ⟹ P (Suc i) v''"* **using** *step* **by** *blast*
  qed
  with *run* **show** *?thesis* **by** *simp*

```
qed

lemma iter'_indexed_type_writes:
  fixes loc :: "nat ⇒ String.literal"
  assumes run: "iter' f v x = Some v'"
    and old: "⋀i j v1 v2. j < i ⟹ f i v1 = Some v2
      ⟹ accessTypeStore (loc j) v2 = accessTypeStore (loc j) v1"
    and new: "⋀i v1 v2. f i v1 = Some v2 ⟹ accessTypeStore (loc i) v2 = Some tp"
  shows "∀j<x. accessTypeStore (loc j) v' = Some tp"
proof (rule iter'_indexed_invariant[OF run, where P = "λi m. ∀j<i. accessTypeStore (loc j) m = Some
tp"])
  show "∀j<0. accessTypeStore (loc j) v = Some tp" by simp
next
  fix i v1 v2
  assume IH: "∀j<i. accessTypeStore (loc j) v1 = Some tp"
    and step: "f i v1 = Some v2"
  show "∀j<Suc i. accessTypeStore (loc j) v2 = Some tp"
  proof (intro allI impI)
    fix j
    assume j_lt: "j < Suc i"
    show "accessTypeStore (loc j) v2 = Some tp"
    proof (cases "j < i")
      case True
      then have "accessTypeStore (loc j) v1 = Some tp"
        using IH by blast
      moreover have "accessTypeStore (loc j) v2 = accessTypeStore (loc j) v1"
        using old[OF True step] by simp
      ultimately show ?thesis
        by simp
    next
      case False
      then have "j = i"
        using j_lt by auto
      then show ?thesis
        using new[OF step] by simp
    qed
  qed
qed

lemma iter_ind:
  assumes "iter f v 0 = v ⟹ P 0 v"
    "⋀x v''. (∃v'. iter f v x = v' ∧ P x v' ∧ f x v' = v'') ⟹ iter f v (Suc x) = v'' ⟹ P (Suc x)
v''"
  shows "iter f v x = v' ⟹ P x v'"
proof (induction x arbitrary: v')
  case 0
  then show ?case using assms(1) by fastforce
next
  case (Suc x)
  from Suc.prems obtain v'' where "f (Suc x-1) v'' = v'" and "iter f v x = v''"
    using iter.simps[of f v "Suc x"] by (auto split: option.split_asm)
  then have "f x v'' = v'" and "iter f v x = v''" by simp+
  then show ?case using assms(2)[OF _ Suc.prems] Suc.IH by blast
qed

lemma iter_indexed_type_writes:
  fixes loc :: "nat ⇒ String.literal"
  assumes run: "iter f v x = v'"
    and old: "⋀i j v1 v2. j < i ⟹ f i v1 = v2
      ⟹ accessTypeStore (loc j) v2 = accessTypeStore (loc j) v1"
    and new: "⋀i v1 v2. f i v1 = v2 ⟹ accessTypeStore (loc i) v2 = Some tp"
  shows "∀j<x. accessTypeStore (loc j) v' = Some tp"
proof (rule iter_ind[OF _ _ run])
  show "iter f v 0 = v ⟹ ∀j<0. accessTypeStore (loc j) v = Some tp"
```

```
    by simp
next
  fix i v2
  assume "∃v1. iter f v i = v1 ∧ (∀j<i. accessTypeStore (loc j) v1 = Some tp) ∧ f i v1 = v2"
  then obtain v1 where IH: "∀j<i. accessTypeStore (loc j) v1 = Some tp"
    and step: "f i v1 = v2" by blast
  show "iter f v (Suc i) = v2 ⟹ ∀j<Suc i. accessTypeStore (loc j) v2 = Some tp"
  proof (intro allI impI)
    fix j
    assume j_lt: "j < Suc i"
    show "accessTypeStore (loc j) v2 = Some tp"
    proof (cases "j < i")
      case True
      then have "accessTypeStore (loc j) v1 = Some tp"
        using IH by blast
      moreover have "accessTypeStore (loc j) v2 = accessTypeStore (loc j) v1"
        using old[OF True step] by simp
      ultimately show ?thesis
        by simp
    next
      case False
      then have "j = i"
        using j_lt by auto
      then show ?thesis
        using new[OF step] by simp
    qed
  qed
qed

lemma updateTypedStore_root_type[simp]:
  "accessTypeStore loc (updateTypedStore loc v tp m) = Some tp"
  unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
  by simp

end
```

3 Locations and sublocations (TypeSafe_Hashing_Subs)

```
theory TypeSafe_Hashing_Subs
  imports "Solidity.Solidity_Main" "HOL-Library.Sublist"
begin
```

3.1 Hashing lemmas

```
fun location_depth :: "location ⇒ nat" where
  "location_depth s = length (filter (λx. x = CHR ''.'') (String.explode s))"

lemma explode_STR_dot: "String.explode (STR ''.'') = [CHR ''.'']"
  using explode_dot by auto

lemma location_depth_hash_property:
  "location_depth (hash ls suf) = location_depth ls + location_depth suf + 1"
  by (simp add: hash_explode add_ac)

lemma hash_adds_depth:
  assumes "CHR ''.'' ∉ set (String.explode suf) "
  shows "location_depth ((hash ls suf)) = location_depth (ls) + 1"
proof -
  have "location_depth suf = 0" using assms filter_False length_0_conv
    by (metis (full_types, lifting) location_depth.simps)
  then show "location_depth (hash ls suf) = location_depth ls + 1 " using location_depth_hash_property
by presburger
qed
```

Hashing a location with any suffix yields a different location.

```
lemma hash_inequality: "∀suffix. (hash ls suffix) ≠ ls "
proof (intro allI notI)
  fix suffix
  assume "hash ls suffix = ls"
  then have "location_depth ls = location_depth ls + location_depth suffix + 1"
    using location_depth_hash_property[of ls suffix] by simp
  then show False by arith
qed

lemma hash_version_inequality: "∀suffix. (hash_version ls suffix) ≠ ls "
proof
  fix suffix
  have " (hash_version ls suffix) = suffix + (STR ''-'' + ls)"
    unfolding hash_version_def by blast
  also have "ls = ls"
    by simp
  ultimately show "hash_version ls suffix ≠ ls"
    using Literal.rep_eq plus_literal.rep_eq literal.explode_inject
    unfolding hash_def nat_explode_def ShowL_nat_def String.implode_def Show_nat_def
    by (metis (no_types, lifting) add.assoc self_append_conv2 snoc_eq_iff_butlast zero_literal.rep_eq)
qed


lemma hash_suffixes_associative: "(hash (hash ls suffix1) suffix2) = (hash ls (hash suffix1 suffix2))"
  unfolding hash_def by (simp add: add.assoc)

lemma not_equal_string:
  assumes "s1≠s2"
  shows "String.explode s1 ≠ String.explode s2"
  by (simp add: assms literal.explode_inject)
```

Distinct prefixes hash to distinct locations.

```
lemma hash_never_equal_prefix:
  assumes "s1 ≠ s2"
  shows "∀iter1. (hash s1 iter1) ≠ hash s2 iter1"
proof intros
  fix iter1
  show "(hash s1 iter1) ≠ hash s2 iter1"
  proof(rule ccontr)
    assume *:"¬ hash s1 iter1 ≠ hash s2 iter1"
    then have **:"hash s1 iter1 = hash s2 iter1" by simp
    then have a10:"String.explode (hash s1 iter1) = String.explode (hash s2 iter1)" by simp
    have a15:"String.explode (hash s1 iter1) = literal.explode iter1 @ literal.explode STR ''.'' @
literal.explode s1" using hash_explode by simp
    have a20:"String.explode (hash s2 iter1) = literal.explode iter1 @ literal.explode STR ''.'' @
literal.explode s2" using hash_explode by simp
    then have "literal.explode s1 = literal.explode s2" using a10 a15 a20 same_append_eq[of
"literal.explode iter1 @ literal.explode STR ''.''" "literal.explode s1" "literal.explode s2"] by simp
    then show False using assms not_equal_string by simp
  qed
qed
```

Distinct base locations hash to distinct locations.

```
lemma hash_never_equal_suffix:
  assumes "s1 ≠ s2"
  shows "∀iter1. (hash iter1 s1) ≠ hash iter1 s2"
proof intros
  fix iter1
  show "hash iter1 s1 ≠ hash iter1 s2"
  proof(rule ccontr)
    assume *:"¬ hash iter1 s1 ≠ hash iter1 s2"
    then have **:"hash iter1 s1 = hash iter1 s2" by simp
```

```
    then have a10:"String.explode (hash iter1 s1) = String.explode (hash iter1 s2)" by simp
    have a15:"String.explode (hash iter1 s1) = literal.explode s1 @ literal.explode STR ''.'' @
literal.explode iter1" using hash_explode by simp
    have a20:"String.explode (hash iter1 s2) = literal.explode s2 @ literal.explode STR ''.'' @
literal.explode iter1" using hash_explode by simp
    then have "literal.explode s1 = literal.explode s2" using a10 a15 a20 same_append_eq by simp
    then show False using assms not_equal_string by simp
  qed
qed

definition LSubPrefL2::"location ⇒ location ⇒ bool"
  where "LSubPrefL2 moreSpecificThanParent parent =
      ((∃i. moreSpecificThanParent = (hash parent i)) ∨ moreSpecificThanParent = parent)"

lemma subPrefCannotBeInt:
  assumes "CHR ''.'' ∉ set(String.explode l)"
  shows "¬(∃i. l = i + (STR ''.'' + stl2))"
proof (rule ccontr)
  assume "¬ (∄i. l = i + (STR ''.'' + stl2))"
  then have a10: "(∃i. l = i + (STR ''.'' + stl2))" by auto
  then show False using assms
    using plus_literal.rep_eq by fastforce
qed

lemma hash_int_prefix:
  assumes "hash destl i ≠ hash destl x"
    and "CHR ''.'' ∉ set(String.explode i)"
    and "CHR ''.'' ∉ set(String.explode x)"
  shows "¬ LSubPrefL2 (hash destl i) (hash destl x)"
  unfolding LSubPrefL2_def
proof
  assume "(∃ia. hash destl i = hash (hash destl x) ia) ∨ hash destl i = hash destl x"
  then show False
  proof
    assume "∃ia. hash destl i = hash (hash destl x) ia"
    then have "∃ia. hash destl i = hash destl (hash x ia)"
      by (simp add: hash_suffixes_associative)
    then have "∃ia. i = (hash x ia)"
      using hash_never_equal_suffix by auto
    then obtain ia where "i = (hash x ia)" by auto
    moreover have "CHR ''.'' ∉ set(String.explode i)" using assms by simp
    ultimately show False using hash_def
      using hash_explode by auto
  next
    assume "hash destl i = hash destl x"
    then show False using assms by simp

  qed

qed


lemma Not_Sub_More_Specific:
  assumes "¬ LSubPrefL2 destl' destl"
  shows "¬ LSubPrefL2 destl' (hash destl x)"
proof
  assume a10:"LSubPrefL2 destl' (hash destl x)"
  have "¬((∃i. destl' = hash destl i) ∨ destl' = destl)" using assms LSubPrefL2_def by simp
  then have *:"(∀i. destl' ≠ hash destl i)" and **:"destl' ≠ destl" by simp+
  have "((∃i. destl' = hash (hash destl x) i) ∨ destl' = hash destl x)" using LSubPrefL2_def[of destl'
"(hash destl x)"] a10 by simp
  then show False
  proof
    assume "∃i. destl' = hash (hash destl x) i"
```

```
    then have "∃i. destl' = hash destl (hash x i)" unfolding hash_def by (simp add: add.assoc)
    then show False using * by simp
  next
    assume "destl' = hash destl x "
    then show False using * by simp
  qed
qed


lemma suffix_eq_harder:
  assumes "b @ s @ a = d @ s @ c"
    and "b ≠ d"
  shows "strict_prefix b d ∨ strict_prefix d b"
  using assms by (metis prefixI prefix_order.less_le prefix_same_cases)

lemma Mutual_NonSub_SpecificNonSub:
  assumes "¬LSubPrefL2 destl' destl ∧ ¬LSubPrefL2 destl destl'"
  shows "¬LSubPrefL2 (hash destl' x) destl"
proof
  assume a10:"LSubPrefL2 (hash destl' x) destl"
  have "¬((∃i. destl' = hash destl i) ∨ destl' = destl)" using assms LSubPrefL2_def by simp
  then have b10:"(∀i. destl' ≠ hash destl i)" and b20:"destl' ≠ destl" by simp+
  have "¬((∃i. destl = hash destl' i) ∨ destl' = destl)" using assms LSubPrefL2_def by simp
  then have b30:"(∀i. destl ≠ hash destl' i)" and b40:"destl' ≠ destl" by simp+
  have "((∃i. hash destl' x = hash destl i) ∨ hash destl' x = destl)" using LSubPrefL2_def[of "(hash
destl' x) " "destl"] a10 by simp
  then show False
  proof
    assume "∃i. hash destl' x = hash destl i"
    then show False
    proof
      fix xa
      assume b50:" hash destl' x = hash destl xa"
      then show False
      proof(cases "x=xa")
        case True
        then show ?thesis using b50 b40 hash_never_equal_prefix by auto
      next
        case False
        then have a10:"String.explode (hash destl' x) = String.explode (hash destl xa)" using b50 by
simp
        have a15:"String.explode (hash destl' x) = literal.explode x @ literal.explode STR ''.'' @
literal.explode destl'" using hash_explode by simp
        have a20:"String.explode (hash destl xa) = literal.explode xa @ literal.explode STR ''.'' @
literal.explode destl" using hash_explode by simp
        then have "strict_prefix (literal.explode x) (literal.explode xa) ∨ strict_prefix
(literal.explode xa) (literal.explode x)"
          using suffix_eq_harder[of "literal.explode x" "literal.explode STR ''.''" "literal.explode
destl'" " literal.explode xa" "literal.explode destl"]
          using a10 a15 a20 False by (simp add: not_equal_string)
        then show ?thesis
        proof
          assume a30:"strict_prefix (literal.explode x) (literal.explode xa)"
          then obtain zs where zs:"literal.explode xa = literal.explode x @ zs" using
strict_prefix_def prefix_def[of " (literal.explode x)" " (literal.explode xa)"]
            by auto
          then have a40:"zs @ literal.explode STR ''.'' @ literal.explode destl = literal.explode STR
''.'' @ literal.explode destl'" using a10 a20 a15 by simp
          then obtain pl where pl2:"literal.explode STR ''.'' @ pl = zs" by (metis Literal.rep_eq a30
append.right_neutral append_Cons
              append_self_conv2 list.exhaust prefix_order.less_imp_not_less strict_prefixI'
strict_prefix_simps(3) zero_literal.rep_eq zs)
          then have a70:"pl @ literal.explode STR ''.'' @ literal.explode destl = literal.explode
destl'" using a40 by auto
```

```
          have "map String.ascii_of pl = pl" using pl2 zs a30 by (metis String.implode_explode_eq
implode.rep_eq map_append same_append_eq)
          then have a90:"(pl @ literal.explode STR ''.'' @ literal.explode destl) = literal.explode
(hash destl (String.implode pl))" using hash_explode[of destl "(String.implode pl)"] by simp
          have a80:"(∀i. literal.explode destl' ≠ literal.explode (hash destl i))" using b10 by (simp
add: not_equal_string)
          then have "∀i. literal.explode destl' ≠ literal.explode i @ literal.explode STR ''.'' @
literal.explode destl" by (simp add: hash_explode)
          then show False using a70 a80 a90 by presburger
        next
          assume a30:"strict_prefix (literal.explode xa) (literal.explode x)"
          then obtain zs where zs:"literal.explode x = literal.explode xa @ zs" using
strict_prefix_def prefix_def[of " (literal.explode xa)" " (literal.explode x)"]
            by auto
          then have a40:"literal.explode STR ''.'' @ literal.explode destl = zs @ literal.explode STR
''.'' @ literal.explode destl'" using a10 a20 a15 by simp
          then obtain pl where pl2:"literal.explode STR ''.'' @ pl = zs" by (metis Literal.rep_eq a30
append.right_neutral append_Cons
              append_self_conv2 list.exhaust prefix_order.less_imp_not_less strict_prefixI'
strict_prefix_simps(3) zero_literal.rep_eq zs)
          then have a70:"pl @ literal.explode STR ''.'' @ literal.explode destl' = literal.explode
destl" using a40 by auto
          have "map String.ascii_of pl = pl" using pl2 zs a30 by (metis String.implode_explode_eq
implode.rep_eq map_append same_append_eq)
          then have a90:"(pl @ literal.explode STR ''.'' @ literal.explode destl') = literal.explode
(hash destl' (String.implode pl))" using hash_explode[of destl' "(String.implode pl)"] by simp
          have a80:"(∀i. literal.explode destl ≠ literal.explode (hash destl' i))" using b30 by (simp
add: not_equal_string)
          then have "∀i. literal.explode destl ≠ literal.explode i @ literal.explode STR ''.'' @
literal.explode destl'" by (simp add: hash_explode)
          then show False using a70 a80 a90 by presburger
        qed
      qed
    qed
  next
    assume " hash destl' x = destl"
    then show False using b30 by auto
  qed
qed

lemma hashesIntSame:
  assumes "hash destl (ShowL_nat i) = y"
    and "hash destl (ShowL_nat i') = y"
  shows "i = i'" using assms
  by (metis Read_Show_nat'_id hash_never_equal_suffix)

lemma hashesInts:
  assumes "hash (hash destl (ShowL_nat i)) z = y"
    and "hash (hash destl (ShowL_nat i')) z' = y"
  shows "i = i'" using assms ShowLNatDot
  by (metis LSubPrefL2_def Read_Show_nat'_id Mutual_NonSub_SpecificNonSub hash_int_prefix
hash_never_equal_suffix)

lemma hashesAssociative:
  assumes "hash (hash destl i) z = y"
    and "hash (hash destl i') z' = y"
    and "CHR ''.'' ∉ set(String.explode i)"
    and "CHR ''.'' ∉ set(String.explode i')"

shows "i = i'" using assms ShowLNatDot
  by (metis LSubPrefL2_def Mutual_NonSub_SpecificNonSub hash_int_prefix hash_never_equal_suffix)

lemma LSubPrefL2_refl: "LSubPrefL2 l l"
  unfolding LSubPrefL2_def by simp
```

lemma *LSubPrefL2_hash: "LSubPrefL2 (hash l k) l"*
unfolding *LSubPrefL2_def* **by** *blast*

lemma *LSubPrefL2_hash_mono:*
assumes *"LSubPrefL2 a b"*
shows *"LSubPrefL2 (hash a k) b"*
using *assms* **unfolding** *LSubPrefL2_def*
by *(metis hash_suffixes_associative)*

lemma *LSubPrefL2_trans:*
assumes *"LSubPrefL2 a b"* **and** *"LSubPrefL2 b c"*
shows *"LSubPrefL2 a c"*
using *assms* **unfolding** *LSubPrefL2_def* **by** *(metis hash_suffixes_associative)*

A location lies below at most one dot-free index of a parent.

lemma *LSubPrefL2_hash_unique:*
assumes *"LSubPrefL2 l (hash p i)"* **and** *"LSubPrefL2 l (hash p j)"*
and *"CHR ''.'' ∉ set (String.explode i)"* **and** *"CHR ''.'' ∉ set (String.explode j)"*
shows *"i = j"*
proof *(rule ccontr)*
assume *ne: "i ≠ j"*
then have *"hash p i ≠ hash p j"* **using** *hash_never_equal_suffix* **by** *blast*
then have *nij: "¬ LSubPrefL2 (hash p i) (hash p j)"*
and *nji: "¬ LSubPrefL2 (hash p j) (hash p i)"*
using *hash_int_prefix assms(3,4)* **by** *metis+*
from *assms(1,2)* **show** *False*
unfolding *LSubPrefL2_def*
proof *(elim disjE exE)*
fix *a b*
assume *"l = hash (hash p i) a" "l = hash (hash p j) b"*
then show *False* **using** *ne hashesAssociative assms(3,4)* **by** *metis*
next
fix *a*
assume *"l = hash (hash p i) a" "l = hash p j"*
then show *False* **using** *nji* **unfolding** *LSubPrefL2_def* **by** *metis*
next
fix *b*
assume *"l = hash p i" "l = hash (hash p j) b"*
then show *False* **using** *nij* **unfolding** *LSubPrefL2_def* **by** *metis*
next
assume *"l = hash p i" "l = hash p j"*
then show *False* **using** *ne hash_never_equal_suffix* **by** *metis*
qed
qed

A location lies below at most one index of an array.

lemma *LSubPrefL2_index_unique:*
assumes *"LSubPrefL2 l (hash p (ShowL$_{nat}$ i))"*
and *"LSubPrefL2 l (hash p (ShowL$_{nat}$ j))"*
shows *"i = j"*
proof *(rule ccontr)*
assume *ne: "i ≠ j"*
then have *"hash p (ShowL$_{nat}$ i) ≠ hash p (ShowL$_{nat}$ j)"*
using *hashesIntSame* **by** *blast*
then have *nij: "¬ LSubPrefL2 (hash p (ShowL$_{nat}$ i)) (hash p (ShowL$_{nat}$ j))"*
and *nji: "¬ LSubPrefL2 (hash p (ShowL$_{nat}$ j)) (hash p (ShowL$_{nat}$ i))"*
using *hash_int_prefix ShowLNatDot* **by** *metis+*
from *assms* **show** *False*
unfolding *LSubPrefL2_def*
proof *(elim disjE exE)*
fix *a b*
assume *"l = hash (hash p (ShowL$_{nat}$ i)) a" "l = hash (hash p (ShowL$_{nat}$ j)) b"*
then show *False* **using** *ne hashesInts* **by** *metis*

```
  next
    fix a
    assume "l = hash (hash p (ShowL_nat i)) a" "l = hash p (ShowL_nat j)"
    then show False using nji unfolding LSubPrefL2_def by metis
  next
    fix b
    assume "l = hash p (ShowL_nat i)" "l = hash (hash p (ShowL_nat j)) b"
    then show False using nij unfolding LSubPrefL2_def by metis
  next
    assume "l = hash p (ShowL_nat i)" "l = hash p (ShowL_nat j)"
    then show False using ne hashesIntSame by metis
  qed
qed

lemma readLintNotEqual:
  assumes "(ReadL_nat suffix')<(ReadL_nat suffixa)"
  shows "suffix' ≠ suffixa" using assms unfolding ReadL_nat_def Read_nat_def nat_implode_def
nat_of_digit_def by blast


lemma checkUIntIncreaseB:
  assumes "(ReadL_int v ≥ 0 ∧ ReadL_int v < 2^b)"
    and "b≤b2"
  shows "(ReadL_int v ≥ 0 ∧ ReadL_int v < 2^b2)"
proof
  show "0 ≤ReadL_int v" using assms by simp
next
  show "ReadL_int v < 2^b2" using assms
    by (metis take_bit_int_eq_self_iff take_bit_int_greater_eq_self_iff take_bit_tightened_less_eq_int)
qed

lemma ShowLnat_ReadLint_inverse:
  assumes "0 ≤ ReadL_int(kv)"
    and "ShowL_int (ReadL_int kv) = kv"
  shows "(ShowL_nat (nat (ReadL_int kv))) = kv"
proof(cases "(ReadL_int kv)")
  case (nonneg n)
  then have a10:"(ReadL_int kv) = Read_int ( String.explode kv)" using ReadL_int_def by (metis comp_apply)
  then have a20:"(ReadL_int kv) = int n" using nonneg by simp
  then have a30: "int n = Read_int ( String.explode kv)" using a10 a20 by simp
  then have a40:"int n = int (Read_nat (String.explode kv))"
  proof(cases "hd (literal.explode kv) = CHR ''-''")
    case headkv:True
    then have a50:"int n = - int (Read_nat (tl (literal.explode kv)))" using a30 Read_int_def by simp
    then have a60:"int n ≤ 0" by simp
    show ?thesis
    proof(cases "n = 0")
      case True
      then have a80:"- int (Read_nat (tl (literal.explode kv))) = 0" using a50 by simp
      have "ShowL_int 0 = String.implode (Show_int 0)" using ShowL_int_def by (metis comp_apply)
      then have a100:"ShowL_int 0 = String.implode ( Show_nat (nat 0))" using Show_int_def by simp
      then have "hd(Show_nat (nat 0)) ≠ CHR ''-''" using Show_nat_def Show_nat_not_neg'' by simp
      then have a120:"hd (literal.explode kv) ≠ hd(Show_nat (nat 0))" using headkv by auto
      then show ?thesis using assms(2) a30 a20 nonneg headkv a50 a80 a100 a120 String.implode_def
using Show_nat_ascii by fastforce
    next
      case False
      then have "n<0" using a60 by simp
      then show ?thesis using nonneg by simp
    qed
  next
    case False
    then show ?thesis using a30 Read_int_def by simp
  qed
```

```
  have "nat (ReadL_int kv) = nat(int (Read_nat (String.explode kv)))" using a40 a30 by (simp add: nonneg)
  then show ?thesis using assms(2) assms(1) comp_apply unfolding ShowL_int_def Show_int_def ShowL_nat_def
by simp
next
  case (neg n)
  then show ?thesis using assms by simp
qed

lemma notNoneUpdate:
  assumes "st' = st(|Stack := updateStore l v (stack st), Gas:=g|)"
  shows "accessStore l (Stack st') = Some v" using assms by fastforce

lemma MemLSubPrefL2_specific_imps_general:
  assumes "LSubPrefL2 (hash p y) (ShowL_nat tloc)"
  shows "LSubPrefL2 p (ShowL_nat tloc)"
proof -
  have a10:"((∃i. hash p y = hash (ShowL_nat tloc) i) ∨ hash p y = (ShowL_nat tloc))"
    using assms LSubPrefL2_def[of "hash p y" "(ShowL_nat tloc)"] by blast
  have a20:" hash p y ≠ (ShowL_nat tloc)" using ShowLNatDot hash_def
    by (metis subPrefCannotBeInt)
  then have a30:"(∃i. hash p y = hash (ShowL_nat tloc) i)" using a10 by auto
  then obtain i where idef:"hash p y = hash (ShowL_nat tloc) i" by blast
  then show ?thesis
  proof(cases "i = y")
    case True
    then have "p = (ShowL_nat tloc)" using a30
      using idef hash_never_equal_prefix by auto
    then show ?thesis
      by (simp add: LSubPrefL2_def)
  next
    case False
    then have a40:"p ≠ (ShowL_nat tloc)" using idef
      using hash_never_equal_suffix by auto
    then have "∃h. hash (ShowL_nat tloc) h = p" using idef False ShowLNatDot hash_def
      by (metis LSubPrefL2_def Mutual_NonSub_SpecificNonSub subPrefCannotBeInt)
    then show ?thesis
      using LSubPrefL2_def by auto
  qed
qed
```

3.2 Miscellaneous lemmas

```
lemma checkSIntIncreaseB:
  assumes "(ReadL_int v ≥ -(2^(b-1)) ∧ ReadL_int v < 2^(b-1))"
    and "b≤b2"
  shows "(ReadL_int v ≥ -(2^(b2-1)) ∧ ReadL_int v < 2^(b2-1))"
proof -
  have "ReadL_int v ≥ -(2^(b2-1))" using assms dual_order.trans by fastforce
  moreover have "ReadL_int v < 2^(b2-1)" using assms
    by (smt (verit, del_insts) diff_le_mono power_increasing)
  ultimately show ?thesis by simp
qed

lemma checkUIntToSInt:
  assumes "(ReadL_int v ≥ 0 ∧ ReadL_int v < 2^b)"
    and "b<b2"
  shows "(ReadL_int v ≥ -(2^(b2-1)) ∧ ReadL_int v < 2^(b2-1))"
proof -
  have "ReadL_int v ≥ -(2^(b2-1))" using assms dual_order.trans by fastforce
  moreover have "ReadL_int v < 2^(b2-1)" using assms
    by (meson checkUIntIncreaseB dec_less_imp_less_eq linorder_not_less)
  ultimately show ?thesis by simp
qed
```

3.3 Additional Hash Helper Lemmas

Hash length and structure properties.

```
lemma hash_contains_dot:
  "CHR ''.'' ∈ set (String.explode (hash a b))"
proof -
  have "String.explode (hash a b) = literal.explode b @ literal.explode STR ''.'' @ literal.explode a"
    using hash_explode by simp
  then show ?thesis using explode_STR_dot by auto
qed

lemma hash_flatten_right:
  "hash a (hash b c) = hash (hash a b) c"
  using hash_suffixes_associative by simp

lemma MemLSubPrefTransitive:
  assumes "LSubPrefL2 loc (ShowL_nat tloc)"
    and "LSubPrefL2 loc p'"
  shows "LSubPrefL2 p' (ShowL_nat tloc)"
proof -
  have a5:"¬(∃x. hash p' x = (ShowL_nat tloc))" using hash_def ShowLNatDot
    by (metis append_Cons explode_dot hash_explode in_set_conv_decomp)
  obtain pre1 where a10: "hash (ShowL_nat tloc) pre1 = loc ∨ loc = (ShowL_nat tloc)" using assms(1)
    using LSubPrefL2_def by auto
  obtain pre2 where a20:"hash p' pre2 = loc ∨ loc = p'" using assms(2)
    using LSubPrefL2_def by auto
  have "((∃i. p' = hash (ShowL_nat tloc) i) ∨ p' = (ShowL_nat tloc))"
  proof(cases "pre1 = pre2")
    case t1:True
    then show ?thesis
    proof(cases "loc = (ShowL_nat tloc)")
      case t2:True
      then show ?thesis
        using a20 a5 by auto
    next
      case False
      then show ?thesis
        using a10 a20 hash_never_equal_prefix t1 by auto
    qed
  next
    case False
    then show ?thesis
      by (metis LSubPrefL2_def a20 a5 assms(1) Mutual_NonSub_SpecificNonSub)
  qed
  then show ?thesis using LSubPrefL2_def[of p' "(ShowL_nat tloc)"] by simp
qed
```

3.4 Stack, calldata, and memory locations are below the top location

```
definition lessThanTopLocs :: "('a, 'v) store_scheme ⇒ bool" where
  "lessThanTopLocs st = ((∀tloc loc. tloc ≥ (Toploc st) ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc st = None)
                          ∧ (∀loc y. accessStore loc st = Some y ⟶ (∃tloc. tloc<(Toploc st) ∧
LSubPrefL2 loc (ShowL_nat tloc))))"


end
```

4 Type consistency for value types (TypeSafe_Base_Types)

```
theory TypeSafe_Base_Types
  imports TypeSafe_Hashing_Subs
begin
```

4.1 Ensuring that String *valuetype* conform to their associated datatype *Valuetypes.types*

```
primrec typeCon :: "types ⇒ valuetype ⇒ bool"
  where "typeCon (TSInt x) v = checkSInt x v"
  | "typeCon (TUInt x) v = checkUInt x v"
  | "typeCon (TBool) v = checkBool v"
  | "typeCon (TAddr) v = checkAddress v"


lemma typeConNoDots:
  assumes "typeCon t v"
  shows "CHR ''.'' ∉ set(String.explode v)"
proof(cases t)
  case (TSInt x1)
  then show ?thesis using assms typeCon.simps(1) ShowLIntDot unfolding checkSInt_def by metis
next
  case (TUInt x2)
  then show ?thesis using assms typeCon.simps(2) ShowLIntDot unfolding checkUInt_def by metis
next
  case TBool
  have "CHR ''.'' ∉ set (literal.explode STR ''True'')" by eval
  moreover have "CHR ''.'' ∉ set (literal.explode STR ''False'')" by eval
  ultimately show ?thesis using assms typeCon.simps(3) unfolding checkBool_def
    using TBool by auto
next
  case TAddr
  then show ?thesis using assms typeCon.simps(4) checkAddress_def by auto
qed

lemma typeCon_no_sublocation_prefix:
  assumes "typeCon x1 i"
  assumes "hash destl i ≠ hash destl x"
  shows "¬ LSubPrefL2 (hash destl i) (hash destl x)"
  unfolding LSubPrefL2_def
proof
  assume "(∃ia. hash destl i = hash (hash destl x) ia) ∨ hash destl i = hash destl x"
  then show False
  proof
    assume "∃ia. hash destl i = hash (hash destl x) ia"
    then have "∃ia. hash destl i = hash destl (hash x ia)"
      by (simp add: hash_suffixes_associative)
    then have "∃ia. i = (hash x ia)"
      using hash_never_equal_suffix by auto
    then obtain ia where "i = (hash x ia)" by auto
    moreover have "CHR ''.'' ∉ set(String.explode i)"
    proof(cases x1)
      case (TSInt x1)
      then show ?thesis using typeCon.simps(1)[of x1 i] assms unfolding checkSInt_def using
ShowLIntDot by metis
    next
      case (TUInt x2)
      then show ?thesis using typeCon.simps(2)[of x2 i] assms unfolding checkUInt_def using
ShowLIntDot by metis
    next
      case TBool
      then have a10:"i = STR ''True'' ∨ i = STR ''False''" using typeCon.simps(3)[of i] assms unfold-
ing checkBool_def by simp
      have "CHR ''.'' ∉ set (literal.explode STR ''True'') ∧ CHR ''.'' ∉ set (literal.explode STR
''False'')" by eval
      then show ?thesis using a10 by auto
    next
      case TAddr
      then show ?thesis using typeCon.simps(4)[of i] assms unfolding checkAddress_def by simp
    qed
```

```
    ultimately show False using hash_def
      using hash_explode by auto
  next
    assume "hash destl i = hash destl x"
    then show False using assms by simp
  qed
qed

lemma addBalance_Bal:
  assumes "addBalance ad val acc = Some acc'"
  shows "Bal (acc' ad) = ShowL_int (ReadL_int (Bal (acc ad)) + ReadL_int val)"
  using assms unfolding addBalance_def by (auto simp: Let_def split: if_split_asm)

lemma subBalance_Bal:
  assumes "subBalance ad val acc = Some acc'"
  shows "Bal (acc' ad) = ShowL_int (ReadL_int (Bal (acc ad)) - ReadL_int val)"
  using assms unfolding subBalance_def by (auto simp: Let_def split: if_split_asm)
```

A successful transfer keeps every balance a valid *uint256*.

```
lemma transfer_Bal_typeCon:
  assumes "transfer ads addr val acc = Some acc'"
    and "∀a. typeCon (TUInt b256) (Bal (acc a))"
  shows "typeCon (TUInt b256) (Bal (acc' a))"
proof -
  from assms(1) obtain acc'' where sub: "subBalance ads val acc = Some acc''"
    and add: "addBalance addr val acc'' = Some acc'"
    by (auto simp: transfer_def split: option.splits)
  have "∀ad. 0 ≤ ReadL_int (Bal (acc ad)) ∧ ReadL_int (Bal (acc ad)) < 2 ^ 256"
    using assms(2) by (simp add: checkUInt_def)
  then have lim: "0 ≤ ReadL_int (Bal (acc' a)) ∧ ReadL_int (Bal (acc' a)) < 2 ^ 256"
    using transfer_limit[OF assms(1)] by blast
  have canon: "ShowL_int (ReadL_int (Bal (acc' a))) = Bal (acc' a)"
  proof (cases "a = addr")
    case True
    then show ?thesis using addBalance_Bal[OF add] Read_ShowL_id by simp
  next
    case False
    then have eq: "Bal (acc' a) = Bal (acc'' a)" using addBalance_eq[OF add] by simp
    show ?thesis
    proof (cases "a = ads")
      case True
      then show ?thesis using eq subBalance_Bal[OF sub] Read_ShowL_id by simp
    next
      case False
      then have "Bal (acc a) = Bal (acc'' a)" using subBalance_eq[OF sub] by metis
      then show ?thesis using eq spec[OF assms(2), of a] by (simp add: checkUInt_def)
    qed
  qed
  show ?thesis using lim canon by (simp add: checkUInt_def)
qed

lemma transfer_subRead:
  assumes "transfer ads addr val acc = Some acc'"
    and "addr ≠ ads"
    and "typeCon (TUInt b256) val"
    and "typeCon (TUInt b256) (Bal (acc ads))"
  shows "(Bal (acc' ads)) = ShowL_int(ReadL_int (Bal (acc ads)) - ReadL_int val)"
proof -
  from assms(1) obtain acc''
    where *: "subBalance ads val acc = Some acc''"
      and **: "addBalance addr val acc'' = Some acc'" by (simp add: subBalance_def transfer_def
split:if_split_asm)

  then have "ReadL_int (Bal (acc'' ads)) = ReadL_int (Bal (acc ads)) - ReadL_int val" using
```

```
subBalance_sub[OF *] by simp
  then have "(Bal (acc'' ads)) = ShowL_int (ReadL_int (Bal (acc ads)) - ReadL_int val)" using * unfolding
subBalance_def
    using "*" subBalance_val1 subBalance_val2 by auto
  moreover from assms(2) have "(Bal (acc' ads)) = (Bal (acc'' ads))" using addBalance_eq[OF **] by
simp
  ultimately show ?thesis using Read_ShowL_id assms by simp
qed

lemma transfer_addRead:
  assumes "transfer ads addr val acc = Some acc'"
    and "addr ≠ ads"
    and "typeCon (TUInt b256) val"
    and "typeCon (TUInt b256) (Bal (acc addr))"
  shows "(Bal (acc' addr)) = ShowL_int(ReadL_int (Bal (acc addr)) + ReadL_int val)"
proof -
  from assms(1) obtain acc''
    where *: "subBalance ads val acc = Some acc''"
      and **: "addBalance addr val acc'' = Some acc'" by (simp add: subBalance_def transfer_def
split:if_split_asm)
  have ***:"(Bal (acc'' addr)) = (Bal (acc addr))"
    using "*" assms(2) subBalance_eq by presburger
  then have "ReadL_int (Bal (acc' addr)) = ReadL_int (Bal (acc'' addr)) + ReadL_int val" using
addBalance_add[OF **] by simp
  then have "(Bal (acc' addr)) = ShowL_int (ReadL_int (Bal (acc addr)) + ReadL_int val)" using * ** ***
unfolding addBalance_def
    by (metis addBalance_Bal[OF **] ***)
  then show ?thesis by simp
qed

lemma transfer_sameRead:
  assumes "transfer ads addr val acc = Some acc'"
    and "addr = ads"
    and "typeCon (TUInt b256) val"
    and "typeCon (TUInt b256) (Bal (acc ads))"
  shows "(Bal (acc' ads)) = (Bal (acc ads))"
proof -
  from assms(1) obtain acc''
    where *: "subBalance ads val acc = Some acc''"
      and **: "addBalance addr val acc'' = Some acc'" by (simp add: subBalance_def transfer_def
split:if_split_asm)
  then have "ReadL_int (Bal (acc ads)) = ReadL_int (Bal (acc' ads))" using transfer_same[OF assms(1) ]
assms(2) by simp
  moreover have "(Bal (acc'' ads)) = ShowL_int (ReadL_int (Bal (acc ads)) - ReadL_int val)" using *
subBalance_def
    by (metis subBalance_Bal[OF *])
  moreover from ** have ***:"ReadL_int (Bal (acc' ads)) = ReadL_int (Bal (acc'' ads)) + ReadL_int val"
using addBalance_add assms(2) by simp
  moreover have "(Bal (acc' addr)) = ShowL_int (ReadL_int (Bal (acc'' addr)) +
      ReadL_int val)" using * ** *** assms(2) unfolding addBalance_def
    by (metis addBalance_Bal[OF **])
  ultimately show ?thesis using Read_ShowL_id assms
    by (simp add: checkUInt_def)
qed
```

4.2 Creating Signed and Unsigned integers

```
lemma checkSIntCreate:
  shows "checkSInt (b) (createSInt b v)" unfolding checkSInt_def using STR_is_int_ShowL Show_ReadL_id
createSInt_def createSInt_greater createSInt_less by force

lemma checkUIntCreate:
  shows "checkUInt (b) (createUInt b v)" unfolding checkUInt_def Read_ShowL_id createUInt_def by (simp
)
```

4.3 Compatible types will both satisfy the consistency of a value

If two types are compatible and a value is consistent with the first, it is consistent with the second.

```
lemma SameCompTypeCon:
  assumes "comp t t'"
    and "typeCon t v"
  shows "typeCon t' v"
proof (cases t)
  case (TSInt x1)
  then obtain x2 where "t' = TSInt x2 ∧ (bits.to_nat x1 ≤ bits.to_nat x2)" using assms(1) by (cases
t'; auto)
  then show ?thesis using assms(2) typeCon.simps(1)[of x1 v ] typeCon.simps(1)[of x2 v ] TSInt
checkSIntIncreaseB checkSInt_def by (cases t'; metis)
next
  case (TUInt x2)
  then obtain x3 where "(t' = TUInt x3 ∧ (bits.to_nat x2 ≤ bits.to_nat x3))
                        ∨ (t' = TSInt x3 ∧ (bits.to_nat x2 < bits.to_nat x3))" using assms(1) by
(cases t'; auto)
  then show ?thesis using assms(2) typeCon.simps(2) typeCon.simps(1) TUInt checkUIntIncreaseB
checkUIntToSInt checkSInt_def checkUInt_def by (cases t'; metis)
next
  case TBool
  then have "t' = t" using comp.elims(2) assms(1) by auto
  then show ?thesis using assms by simp
next
  case TAddr
  then have "t' = t" using comp.elims(2) assms(1) by auto
  then show ?thesis using assms by simp
qed
```

4.4 Initial values

When a new location is created in Memory, Storage or calldata, or a new variable is created then a default initial value is used. These values must satisfy *typeCon*, which is what the following lemma demonstrates.

```
lemma ivalTypeCon:
  assumes "ival t' = v'"
  shows "typeCon t' v'"
proof (cases t')
  case (TSInt x1)
  have a1:"ShowL_int 0 = STR ''0''" by Solidity_Symbex.solidity_symbex
  then have a10:"ReadL_int (STR ''0'') = (0::int)" by Solidity_Symbex.solidity_symbex
  then show ?thesis using checkSInt_def[of x1 "STR ''0''"] TSInt assms a1 by simp
next
  case (TUInt x2)
  have a1:"ShowL_int 0 = STR ''0''" by eval
  then have a10:"ReadL_int (STR ''0'') = (0::int)" by eval
  then show ?thesis using checkUInt_def[of x2 "STR ''0''"] TUInt assms a1 by simp
next
  case TBool
  have "ShowL_bool False = STR ''False''" by eval
  then show ?thesis using TBool checkBool_def assms by auto
next
  case TAddr
  then have "v' = STR ''0x0000000000000000000000000000000000000000''" using ival.simps(4) assms by
simp
  moreover have "(CHR ''.'' ∉ set (literal.explode STR ''0x0000000000000000000000000000000000000000''))"
by eval
  ultimately show ?thesis using typeCon.simps(4) TAddr checkAddress_def by auto
qed
```

4.5 Convert is typesafe

Conversion between types preserves type consistency. If the string value was consistent with the initial type and a successful conversion is performed then the result must be consistent with the new type.

```
lemma typeSafeConvert:
  assumes "typeCon t2 v"
    and "convert t2 t' v = Some(v)"
  shows "typeCon t' v"
proof (cases "t2 = t'")
  case True
  then show ?thesis using assms by simp
next
  case False
  then show ?thesis
  proof(cases t2)
    case t2x1:(TSInt x1)
    then show ?thesis
    proof(cases t')
      case (TSInt x1a)
      then have "bits.to_nat(x1) ≤ bits.to_nat(x1a)" using assms(2) convert.simps(1)[of x1 x1a v] by
(metis TSInt not_Some_eq t2x1)
      then show ?thesis using assms(1) typeCon.simps(1)[of x1 v] typeCon.simps(1)[of x1a v]
checkSIntIncreaseB checkSInt_def
        by (metis TSInt t2x1)
    next
      case (TUInt x2)
      then have "convert (TSInt x1) (TUInt x2) v = None" using convert.simps by simp
      then have False using assms t2x1 TUInt by simp
      then show ?thesis by simp
    next
      case TBool
      then have "convert (TSInt x1) TBool v = None" using convert.simps by simp
      then have False using assms t2x1 TBool by simp
      then show ?thesis by simp
    next
      case TAddr
      then have "convert (TSInt x1) TAddr v = None" using convert.simps by simp
      then have False using assms t2x1 TAddr by simp
      then show ?thesis by simp
    qed
  next
    case t2x2:(TUInt x2)
    then show ?thesis
    proof(cases t')
      case (TSInt x1a)
      then have "bits.to_nat(x2) < bits.to_nat(x1a)" using assms(2) convert.simps(1)[of x2 x1a v] by
(metis convert.simps(3) option.distinct(1) t2x2)
      then show ?thesis using assms(1) typeCon.simps(2)[of x2 v] typeCon.simps(1)[of x1a v]
checkUIntToSInt checkSInt_def checkUInt_def TSInt assms(2) convert.simps(3) t2x2 by force
    next
      case (TUInt x2a)
      then have "bits.to_nat(x2)≤bits.to_nat(x2a)" using assms(2) convert.simps(2)[of x2 x2a v] by
(metis option.discI t2x2)
      then show ?thesis using checkUIntIncreaseB assms t2x2 TUInt typeCon.simps(2) convert.simps(2)
checkUInt_def by auto
    next
      case TBool
      then have "convert (TUInt x2) TBool v = None" using convert.simps by simp
      then have False using assms t2x2 TBool by simp
      then show ?thesis by simp
    next
      case TAddr
      then have "convert (TUInt x2) TAddr v = None" using convert.simps by simp
      then have False using assms t2x2 TAddr by simp
```

```
      then show ?thesis by simp
    qed
  next
    case TBool
    then have "t' ≠ TBool" using False by simp
    then have "convert t2 t' v = None" using convert.simps by (metis TBool types.exhaust)
    then have False using assms by simp
    then show ?thesis by simp
  next
    case TAddr
    then have "t' ≠ TAddr" using False by simp
    then have "convert t2 t' v = None" using convert.simps by (metis TAddr types.exhaust)
    then have False using assms by simp
    then show ?thesis by simp
  qed
qed


end
```

5 Type consistency for memory (TypeSafe_Memory)

```
theory TypeSafe_Memory
  imports TypeSafe_Base_Types
begin

fun extractType :: "mtypes ⇒ mtypes option" where
  "extractType (MTArray len arr) = Some arr"
| "extractType (MTValue v) = None"

definition AllocatedMem_between :: "memoryT ⇒ memoryT ⇒ location set"
  where
    "AllocatedMem_between st st' = ({ShowL_nat l | l. Toploc st ≤ l ∧ l < Toploc st'})
                                  ∪ fset(fmdom (Mapping (st')) - fmdom (Mapping (st)))"

primrec CompMemJustType::"mtypes ⇒ mtypes ⇒ bool"
  where
    "CompMemJustType (MTValue typ) mem = (mem = MTValue typ)"
  | "CompMemJustType (MTArray len arr) mem = (mem = (MTArray len arr) ∨ CompMemJustType arr mem)"

primrec CompMemType :: "memoryT ⇒ nat ⇒ mtypes ⇒ mtypes ⇒ location ⇒ location ⇒ bool"
  where
    "CompMemType mem len (MTValue typ) childtp parentloc childloc = (childtp = (MTValue typ) ∧ (∃i<len.
(hash parentloc (ShowL_nat i)) = childloc))"
  | "CompMemType mem len (MTArray len' arr) childtp parentloc childloc = (∃i<len. ∃l. accessStore (hash
parentloc (ShowL_nat i)) mem = Some (MPointer l) ∧
                                                                    ((l = childloc ∧ (MTArray
len' arr) = childtp) ∨ CompMemType mem len' arr childtp l childloc)
                                                                  )"
```

All memory locations occupied by a value of the given memory type, starting at a base location and following array pointers in the given memory.

```
fun GetAllMemoryLocations :: "mtypes ⇒ location ⇒ memoryT ⇒ location set"
  where
    "GetAllMemoryLocations (MTValue _) base_loc _ = {base_loc}"
  | "GetAllMemoryLocations (MTArray len struct) base_loc mem =
      (⋃i∈{0..<len}.
        (case accessStore (hash base_loc (ShowL_nat i)) mem of
           Some (MPointer ptr) ⇒ {(hash base_loc (ShowL_nat i))} ∪ GetAllMemoryLocations struct ptr mem
         | Some (MValue _) ⇒ {(hash base_loc (ShowL_nat i))} ∪ GetAllMemoryLocations struct (hash
base_loc (ShowL_nat i)) mem
         | None ⇒ {}))"
```

5.1 Memory and calldata conformity

MCon checks that the sub-locations of a memory or calldata structure conform to their types. Starting at a location, it checks all sub-locations. For an array, every element is checked recursively; for a value type, the string stored at the location is checked against its base type.

Locations stored in the *Denvalue* for memory references already denote the target of the pointer, following the convention of *expressions_with_gas.msel*. For example, if a memory array stored at *0.0* has pointer value *7.7*, the environment stores *7.7*. Hence *MCon* does not dereference its location argument.

```
primrec MCon :: "mtypes ⇒ memoryT ⇒ location ⇒ bool"
  where
    "MCon (MTValue typ) msto loc =(case (accessStore loc msto) of
                                   (Some (MValue t)) ⇒ (typeCon typ t)
                                     | (Some(MPointer t)) ⇒ False
                                     | None ⇒ False)"

|"MCon (MTArray len arr) msto loc = (len>0 ∧ (∀ i < len.
                                              (case (accessStore (hash loc (ShowL_nat i)) msto) of
                                               (Some (MPointer loc2)) ⇒
                                                      (case arr of (MTArray len' arr') ⇒ (MCon arr
msto (loc2))
                                                                 | MTValue val ⇒ False)
                                                | Some(MValue val) ⇒
                                                  (case arr of (MTValue typ) ⇒ (MCon arr msto (hash
loc (ShowL_nat i)))
                                                     | (MTArray l a) ⇒ False)
                                                | None ⇒ False))
                                    ∧ (∃p. accessStore loc msto = Some (MPointer p) ∨ accessStore loc
msto = None))"

primrec MConAccessSame:: "location ⇒ location ⇒ mtypes ⇒ memoryT ⇒ memoryT ⇒ bool"
  where
    "MConAccessSame l1 l2 (MTValue t) m1 m2 = (∃val. accessStore l1 m1 = Some (MValue val) ∧ (∃val'.
accessStore l2 m2 = Some (MValue val')))"
  | "MConAccessSame l1 l2 (MTArray len t) m1 m2 = (∀i1<len. ∀i2<len. (∀l. accessStore (hash l1
(ShowL_nat i1)) m1 = Some (MPointer l) ⟶
                                                                    (∃l'. accessStore (hash l2 (ShowL_nat
i2)) m2 = Some (MPointer l')
                                                                    ∧ MConAccessSame l l' t m1 m2)))"
```

The set of all memory locations reachable from the memory references of an environment and stack.

```
fun ReachableMem :: "environment ⇒ stack ⇒ memoryT ⇒ location set"
  where
    "ReachableMem env st mem = (⋃ (t, l) ∈ fset(fmran (Denvalue env)).
         (case l of
           Stackloc loc ⇒
             (case accessStore loc st of
               Some (KMemptr ptr) ⇒
                 (case t of
                   type.Memory struct ⇒ GetAllMemoryLocations struct ptr mem
                    | _ ⇒ {})
               | _ ⇒ {})
            | Storeloc loc ⇒ {}))"


primrec TypedMemSubPref:: "location ⇒ location ⇒ mtypes ⇒ bool"
  where
    "TypedMemSubPref child parent (MTValue t) = (child = parent)"
  | "TypedMemSubPref child parent (MTArray len t) = (∃i<len. TypedMemSubPref child (hash parent
(ShowL_nat i)) t
      ∨ child = (hash parent (ShowL_nat i)))"

primrec TypedMemSubPrefPtrs:: "memoryT ⇒ nat ⇒ mtypes⇒ location ⇒ location ⇒ bool"
  where
```

```
    "TypedMemSubPrefPtrs mem len (MTValue t) parent child = (∃ i<len. (hash parent (ShowL_nat i)) =
child)"
  | "TypedMemSubPrefPtrs mem len (MTArray len' t) parent child = (∃ i<len. ∃ l. accessStore (hash parent
(ShowL_nat i)) mem = Some (MPointer l)
                                                                  ∧ (l = child ∨ TypedMemSubPrefPtrs
mem len' t l child ))"

definition WrittenMem_between :: "memoryT ⇒ memoryT ⇒ location set"
  where
    "WrittenMem_between st st' = {l. accessStore l st ≠ accessStore l (st')}"

definition compMemPtrs :: "stack ⇒ memoryT ⇒ (String.literal, type × denvalue) fmap ⇒ bool" where
  "compMemPtrs st mem denval = (∀ tp1 tp2 l1 l2 stl1 stl2 dloc1.
         (type.Memory tp1, Stackloc l1) |∈| fmran denval ∧ (type.Memory tp2, Stackloc l2) |∈| fmran
denval
         ∧ accessStore l1 st = Some(KMemptr stl1) ∧ accessStore l2 st = Some(KMemptr stl2)
      ⟶ (if stl1 = stl2 then tp1 = tp2 else
           (case tp1 of
             MTValue val ⇒
               (case tp2 of MTValue val2 ⇒ True
                 | MTArray len2 arr2 ⇒ (if TypedMemSubPrefPtrs mem len2 arr2 stl2 stl1 then
CompMemType mem len2 arr2 (MTValue val) stl2 stl1
                                       else True))
            | MTArray len arr ⇒
               (case tp2 of MTValue val2 ⇒ (if TypedMemSubPrefPtrs mem len arr stl1 stl2 then
CompMemType mem len arr (MTValue val2) stl1 stl2
                                         else True)
                 | MTArray len2 arr2 ⇒ (if TypedMemSubPrefPtrs mem len2 arr2 stl2 stl1 then
CompMemType mem len2 arr2 (MTArray len arr) stl2 stl1
                                       else if TypedMemSubPrefPtrs mem len arr stl1 stl2 then
CompMemType mem len arr (MTArray len2 arr2) stl1 stl2
                                       else if (TypedMemSubPrefPtrs mem len arr stl1 dloc1 ∧
TypedMemSubPrefPtrs mem len2 arr2 stl2 dloc1)
                                              then (∃ dt. CompMemType mem len2 arr2 dt stl2 dloc1
                                                                  ∧ CompMemType mem len arr dt
stl1 dloc1
                                                                  )
                                       else True))

)))"

definition compMemPtrs3 :: "stack ⇒ memoryT ⇒ (String.literal, type × denvalue) fmap ⇒ bool" where
  "compMemPtrs3 st mem denval = (∀ tp1 tp2 l1 l2 stl1 stl2 dloc1.
    (type.Memory tp1, Stackloc l1) |∈| fmran denval ∧ (type.Memory tp2, Stackloc l2) |∈| fmran denval
    ∧ accessStore l1 st = Some(KMemptr stl1) ∧ accessStore l2 st = Some(KMemptr stl2)
    ⟶ (if stl1 = stl2 then tp1 = tp2 else
      (case (tp1, tp2) of
          (MTValue val, MTValue val2) ⇒ True
        | (MTValue val, MTArray len2 arr2) ⇒ ¬ (TypedMemSubPrefPtrs mem len2 arr2 stl2 stl1) ∨
CompMemType mem len2 arr2 (MTValue val) stl2 stl1
        | (MTArray len arr, MTValue val2) ⇒ ¬ (TypedMemSubPrefPtrs mem len arr stl1 stl2) ∨
CompMemType mem len arr (MTValue val2) stl1 stl2
        | (MTArray len arr, MTArray len2 arr2) ⇒
            if (TypedMemSubPrefPtrs mem len arr stl1 dloc1 ∧ TypedMemSubPrefPtrs mem len2 arr2 stl2
dloc1)
                then (∃ dt. CompMemType mem len2 arr2 dt stl2 dloc1 ∧ CompMemType mem len arr dt stl1
dloc1 )
            else True)
      )
    )"


fun mtype_size :: "mtypes ⇒ nat" where
  "mtype_size (MTValue _) = 1"
```

```
| "mtype_size (MTArray _ S) = 1 + mtype_size S"
```

5.2 Type relations between arrays and their elements

```
lemma CompMemJustTypes_trns:
  assumes "CompMemJustType tp1 tp2"
    and "CompMemJustType tp1 tp3"
  shows "CompMemJustType tp2 tp3 ∨ CompMemJustType tp3 tp2" using assms
proof(induction tp1 arbitrary:tp2 tp3)
  case (MTArray x1 tp1)
  then show ?case by auto
next
  case (MTValue x)
  then show ?case by simp
qed

lemma CompMemType_SameAccessAllocate:
  assumes "∀loc. accessStore loc m = accessStore loc (snd (allocate m'))"
    and "CompMemType m len t1 t2 l1 l2"
  shows "CompMemType m' len t1 t2 l1 l2" using assms
proof(induction t1 arbitrary:len l1)
  case (MTArray x1 t1)
  then obtain i l where idef:"i<len ∧ accessStore (hash l1 (ShowL_nat i)) m = Some (MPointer l)
                              ∧ (l = l2 ∧ MTArray x1 t1 = t2 ∨ CompMemType m x1 t1 t2 l l2)"
    unfolding CompMemType.simps by auto
  then show ?case
  proof(cases "l = l2")
    case True
    then show ?thesis using idef MTArray.prems
      by (metis (no_types, lifting) CompMemType.simps(2) MTArray.IH allocateSameAccess)
  next
    case False
    then show ?thesis using idef MTArray
      by (metis CompMemType.simps(2) allocateSameAccess)
  qed
next
  case (MTValue x)
  then show ?case by auto
qed

lemma CompMemType_imps_CompMemJustType:
  assumes "CompMemType mem len t1 t2 l1 l2"
  shows "CompMemJustType (MTArray len t1) t2" using assms
proof(induction t1 arbitrary:l1 len)
  case (MTArray x1 t2')
  then show ?case
  proof(cases "t2 = MTArray x1 t2'")
    case True
    then show ?thesis by simp
  next
    case False
    then have a10:"(∃i<len. ∃l. accessStore (hash l1 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l =
        l2 ∧ MTArray x1 t2' = t2 ∨ CompMemType mem x1 t2' t2 l l2))"
      using MTArray(2) CompMemType.simps(2)[of mem len x1 t2' t2 l1 l2] by simp

    have "CompMemJustType t2' t2"
      using False MTArray.IH a10 by auto
    then show ?thesis using False by simp
  qed
next
  case (MTValue x)
  then show ?case using CompMemJustType.simps(1) CompMemType.simps(1) by simp
qed
```

```
lemma CompMemType_imps_TypedMemSubPrefPtrs:
  assumes "CompMemType mem len tp tc lp lc"
  shows "TypedMemSubPrefPtrs mem len tp lp lc" using assms
proof(induction tp arbitrary:lp len)
  case (MTArray x1 tp)
  then show ?case
    using MTArray.IH by auto
next
  case (MTValue x)
  then show ?case by simp
qed

lemma CompMemTypes_asc:
  assumes "CompMemJustType tp1 tp2"
    and "CompMemJustType tp2 tp3"
  shows "CompMemJustType tp1 tp3" using assms
proof(induction tp1)
  case (MTArray x1 tp1)
  then show ?case by auto
next
  case (MTValue x)
  then show ?case by simp
qed

lemma MConArrayPointers:
  assumes "MCon (MTArray len (MTArray len' t)) v' loc"
    and "len >0"
    and "i < len"
  obtains val where "accessStore (hash loc (ShowL_nat i)) v' = Some (MPointer val)"
proof -
  have a10:"∀ i<len.
             case accessStore (hash loc (ShowL_nat i)) v' of None ⇒ False
             | Some (MValue val) ⇒
                 (case MTArray len' t of MTValue typ ⇒ MCon (MTArray len' t) v' (hash loc (ShowL_nat
i))
                  | _ ⇒ False)
             | Some (MPointer loc2) ⇒
                 (case MTArray len' t of MTArray len'a arr' ⇒ MCon (MTArray len' t) v' loc2
                 | MTValue Types ⇒ False)"
    using MCon.simps(2)[of len "MTArray len' t" v' loc] assms(1) by simp
  show ?thesis
  proof(cases "accessStore (hash loc (ShowL_nat i)) v'")
    case None
    then show ?thesis using a10 assms by auto
  next
    case (Some a)
    then show ?thesis
    proof(cases a)
      case (MValue x1)
      then show ?thesis using a10 assms Some by auto
    next
      case MPointer
      then show ?thesis using Some that by auto
    qed
  qed
qed


lemma MCon_array_decrement:
  assumes "MCon (MTArray len t) v' loc"
    and "len -1 > 0"
  shows "MCon (MTArray (len-1) t) v' loc" using assms MCon.simps by simp

lemma MConTypeSplitingArray:
```

```
  assumes "∀(i::nat)<(x1::nat).
             (case accessStore (hash stl1 (ShowL_nat i)) mem of
                Some (MPointer loc2) =>
                    (case tp1 of MTArray len' arr' => MCon tp1 mem loc2
                               | MTValue tps => False)
              | Some (MValue val) =>
                    (case tp1 of (MTArray n mt) => False
                               | MTValue tps => MCon tp1 mem (hash stl1 (ShowL_nat i)))
              | None => False)"
    and "∃a b. tp1 = MTArray a b"
    and "(i::nat)<x1"
  shows "∃p. accessStore (hash stl1 (ShowL_nat i)) mem = Some (MPointer p)"
proof(cases "accessStore (hash stl1 (ShowL_nat i)) mem")
  case None
  then show ?thesis using assms(1) assms(3) by auto
next
  case (Some a)
  then show ?thesis
  proof(cases a)
    case (MValue x1)
    then show ?thesis using assms Some by auto
  next
    case (MPointer x2)
    then show ?thesis using assms Some by auto
  qed
qed

lemma MConSubTypes:
  assumes "MCon (MTArray len t') mem loc"
  shows "∀t. CompMemJustType t' t ⟶ (¬MCon t mem loc)" using assms
proof(induction t' arbitrary: len loc)
  case (MTArray x1 t')
  show ?case
  proof intros
    fix t
    assume *:"CompMemJustType (MTArray x1 t') t"
    then have a5:"t = MTArray x1 t' ∨ CompMemJustType t' t" by (cases t) auto

    have a10:" ∀i<len.
             case accessStore (hash loc (ShowL_nat i)) mem of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x1 t' of MTArray n mtypes ⇒ False
                                    | MTValue typ ⇒ MCon (MTArray x1 t') mem (hash loc (ShowL_nat
i)))
             | Some (MPointer loc2) ⇒ (case MTArray x1 t' of MTArray len' arr' ⇒ MCon (MTArray x1 t')
mem loc2
                                    | MTValue Types ⇒ False)"
      using MTArray(2) MCon.simps(2)[of len "MTArray x1 t'" mem loc] by simp
    then obtain p' i where p'def:" i<len ∧ accessStore (hash loc (ShowL_nat i)) mem = Some (MPointer
p')"
      by (metis MConArrayPointers MTArray.prems(1) MCon.simps(2))
    then have a20:"MCon (MTArray x1 t') mem p'" using a10 by fastforce
    then have a30:"∀t. CompMemJustType t' t ⟶ ¬ MCon t mem p'" using MTArray.IH by auto

    then show "¬ MCon t mem loc"
    proof(cases "t = MTArray x1 t'")
      case True
      show ?thesis
      proof
        assume ***:"MCon t mem loc"
        then have a90:"∀i<x1.
             (case accessStore (hash loc (ShowL_nat i)) mem of None ⇒ False
                  | Some (MValue val) ⇒ (case t' of MTArray n mtypes ⇒ False | MTValue typ ⇒ MCon t'
mem (hash loc (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case t' of MTArray len' arr' ⇒ MCon t' mem loc2 | MTValue
```

```
Types ⇒ False))"
          using MCon.simps(2)[of x1 t' mem loc ] using True by auto
        then obtain i' where i'Def:"i'<x1 ∧ i' <len"
          using "*" MTArray.prems <MCon t mem loc>
          using True by fastforce
        then obtain p'' where p''def: "accessStore (hash loc (ShowL_nat i')) mem = Some (MPointer p'')"
using a10 p'def
            MConTypeSplitingArray[of len "MTArray x1 t'" mem loc i'] by auto
        then have "MCon t' mem p''" using a90 i'Def
          by (metis (no_types, lifting) mtypes.exhaust mtypes.simps(5) mtypes.simps(6)
memoryvalue.simps(6) Option.option.simps(5))
        moreover have "MCon (MTArray x1 t') mem p'''" using p''def i'Def a10 by auto
        ultimately show False using a20
          using "*" MTArray.prems <MCon t mem loc>
          by (metis MTArray.IH CompMemJustType.simps(1) CompMemJustType.simps(2) extractType.elims)
      qed
    next
      case False
      then have "CompMemJustType t' t" using a5 by auto
      then have "¬ MCon t mem p'" using a30 by auto
      show ?thesis
      proof
        assume ***:"MCon t mem loc"
        then show False
        proof(cases t)
          case mta:(MTArray x11 x12)
          then have c10:"∀ i<x11.
              (case accessStore (hash loc (ShowL_nat i)) mem of None ⇒ False
               | Some (MValue val) ⇒ (case x12 of MTArray n mtypes ⇒ False | MTValue typ ⇒ MCon x12
mem (hash loc (ShowL_nat i)))
               | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12 mem loc2 | MTValue
Types ⇒ False))"
            using *** MCon.simps(2)[of x11 x12 mem loc] by simp
          then obtain i''' where i'''def:"i'''<x11 ∧ i'''<len"
            by (metis "***" mta MTArray.prems neq0_conv MCon.simps(2))
          then obtain p'' where p''def: "accessStore (hash loc (ShowL_nat i''')) mem = Some (MPointer
p'')"
            using a10 p'def MConTypeSplitingArray[of len "MTArray x1 t'" mem loc i'''] by auto
          then have "MCon (MTArray x1 t') mem p'''" using a10 i'''def by fastforce
          have "MCon x12 mem p''" using i'''def p''def c10 mta
            by (metis (no_types, lifting) mtypes.exhaust mtypes.simps(5) mtypes.simps(6)
memoryvalue.simps(6) Option.option.simps(5))
          then show ?thesis using MCon.simps(2)[of x11 x12 mem loc]
            by (metis CompMemTypes_asc MTArray.IH <CompMemJustType t' t> <MCon (MTArray x1 t') mem p''>
mta
                CompMemJustType.simps(1) CompMemJustType.simps(2) extractType.cases)
        next
          case (MTValue x2)
          then show ?thesis using p'def *** MCon.simps(1)[of x2 mem loc]
            using MTArray.prems by force
        qed

      qed
    qed
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix t
    assume *:"CompMemJustType (MTValue x) t "
    then have **:"t = MTValue x" by simp
    moreover have "∀ i<len.
            case accessStore (hash loc (ShowL_nat i)) mem of None ⇒ False
```

```
            | Some (MValue val) ⇒ (case MTValue x of MTArray n mtypes ⇒ False | MTValue typ ⇒ MCon
(MTValue x) mem (hash loc (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTValue x of MTArray len' arr' ⇒ MCon (MTValue x) mem loc2
| MTValue Types ⇒ False)"
      using MTValue MCon.simps(2)[of len "MTValue x" mem loc] by auto
    then show "¬ MCon t mem loc" using MTValue **
      by (metis memoryvalue.simps(6) Option.option.simps(4) Option.option.simps(5) MCon.simps(1)
MCon.simps(2))
  qed
qed

lemma MConPtrsMustBeSubLocs:
  assumes "MCon (MTArray len t') mem loc"
  shows "∀p i. i<len ∧ accessStore (hash loc (ShowL_nat i)) mem = Some (MPointer p) ⟶ p ≠ loc"
proof intros
  fix i p
  assume *:"i < len ∧ accessStore (hash loc (ShowL_nat i)) mem = Some (MPointer p)"
  then have a10:"(case t' of MTArray len' arr' ⇒ MCon t' mem p | MTValue Types ⇒ False)"
    using assms MCon.simps(2)[of len t' mem loc] by auto
  then have "MCon t' mem p" using * by (cases t'; simp)
  then show "p≠loc" using assms MConSubTypes[of len t' mem loc]
    by (metis CompMemJustType.simps(1) CompMemJustType.simps(2) extractType.cases)
qed

lemma MConPtrsMustBeSubLocs_CompMemType:
  assumes "MCon (MTArray len arr) mem loc"
  shows "∀p t. MCon t mem p ∧ CompMemType mem len arr t loc p ⟶ p ≠ loc" using assms
proof (induction arr arbitrary:len loc)
  case (MTArray x1 arr)
  show ?case
  proof(intros)
    fix p t
    assume *:"MCon t mem p ∧ CompMemType mem len (MTArray x1 arr) t loc p"

    show "p ≠ loc"
    proof
      assume **:"p = loc"
      have "(∃i<len. ∃l. accessStore (hash loc (ShowL_nat i)) mem = Some (MPointer l)
            ∧ (l = p ∧ MTArray x1 arr = t ∨ CompMemType mem x1 arr t l p))"
        using * CompMemType.simps(2)[of mem len x1 arr t loc p] by auto
      then obtain i l where idef:"i<len ∧ accessStore (hash loc (ShowL_nat i)) mem = Some (MPointer l)
                                  ∧ (l = p ∧ MTArray x1 arr = t ∨ CompMemType mem x1 arr t l p)" by
blast
      then have "(case accessStore (hash loc (ShowL_nat i)) mem of None ⇒ False
              | Some (MValue val) ⇒ (case MTArray x1 arr of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                  (MTArray x1 arr) mem (hash loc (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case MTArray x1 arr of MTArray len' arr' ⇒ MCon (MTArray x1
arr) mem loc2 | MTValue Types ⇒ False))
            ∧ ((∃p. accessStore loc mem = Some (MPointer p)) ∨ accessStore loc mem = None)"
        using MTArray(2) MCon.simps(2)[of len "MTArray x1 arr" mem loc] by metis
      then have "MCon (MTArray x1 arr) mem l" using idef by force
      show False
      proof(cases "l = p ∧ MTArray x1 arr = t")
        case True
        then show ?thesis
          using "**" idef
          using MConPtrsMustBeSubLocs MTArray.prems by blast
      next
        case False
        then have "CompMemType mem x1 arr t l p" using idef by auto
        then have "p ≠ l"
          using "**" MConPtrsMustBeSubLocs MTArray.prems MConSubTypes[of len "(MTArray x1 arr)" mem
loc]
          using idef by blast
```

```
        then show ?thesis
          using "*" "**" CompMemType_imps_CompMemJustType MConSubTypes MTArray.prems <CompMemType mem
x1 arr t l p> by blast
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case by fastforce
qed


lemma BothMConImpsNotCompMemType:
  assumes "MCon (MTArray len arr) mem p''"
    and "MCon t' mem p''"
  shows "¬CompMemType mem len arr t' p'' p''" using assms
proof(induction arr arbitrary:len p'')
  case (MTArray x1 arr)
  then show ?case
    by (meson MConPtrsMustBeSubLocs_CompMemType)
next
  case (MTValue x)
  then show ?case by fastforce
qed

lemma MCon_imps_Some:
  assumes "MCon t' mem p'"
  shows "∃x i. accessStore p' mem = Some x ∨ accessStore (hash p' (ShowL_nat i)) mem = Some x"
proof(cases t')
  case (MTArray x11 x12)
  then have a10:"∀i<x11.
             (case accessStore (hash p' (ShowL_nat i)) mem of None ⇒ False
              | Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 mem
(hash p' (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12 mem loc2 | MTValue
Types ⇒ False))"
    using assms MCon.simps(2)[of x11 x12 "mem" p'] by simp
  then obtain i where a20:"i<x11 "
    using MTArray assms by fastforce
  then show ?thesis
  proof(cases " accessStore (hash p' (ShowL_nat i)) mem")
    case None
    then show ?thesis using a10 a20 by auto
  next
    case (Some a)
    then show ?thesis
      by auto
  qed
next
  case (MTValue x2)
  then have a10:"(case accessStore p' mem of None ⇒ False | Some (MValue t) ⇒ typeCon x2 t | Some
(MPointer t) ⇒ False)"
    using assms(1) MCon.simps(1)[of x2 "mem" p'] by auto
  then show ?thesis
  proof(cases "accessStore p' mem")
    case None
    then show ?thesis using a10 by simp
  next
    case (Some a)
    then show ?thesis by auto
  qed
qed

lemma MCon_subTypes_imps_noPrnt:
```

```
  assumes "MCon (MTArray x t) mem l"
    and "MCon (MTArray x' (MTArray x t)) mem l'"
  shows "¬CompMemType mem x t (MTArray x' (MTArray x t)) l l'" using assms
  using CompMemType_imps_CompMemJustType MConSubTypes by blast

abbreviation example_mem_array_bool::memoryT
  where "example_mem_array_bool ≡
    (|Mapping = fmap_of_list
      [(STR ''1.1.0'', MValue STR ''False''),
       (STR ''0.1.0'', MValue STR ''True''),
       (STR ''1.0'', MPointer STR ''1.0''),
       (STR ''1.0.0'', MValue STR ''False''),
       (STR ''0.0.0'', MValue STR ''True''),
       (STR ''0.0'', MPointer STR ''0.0'')],
     Toploc = 1,
     Typed_Mapping = fmap_of_list
      [(STR ''1.1.0'', MTValue TBool),
       (STR ''0.1.0'', MTValue TBool),
       (STR ''1.0'', MTArray 2 (MTValue TBool)),
       (STR ''1.0.0'', MTValue TBool),
       (STR ''0.0.0'', MTValue TBool),
       (STR ''0.0'', MTArray 2 (MTValue TBool))]|)"

lemma MCon_sub_MTVal_imps_val:
  assumes "MCon (MTArray len (MTValue t)) mm loc"
  shows "∀i<len. ∃val. accessStore (hash loc (ShowL_nat i)) mm = Some (MValue val)"
proof intros
  fix i
  assume "i<len"
  then have a10:"case accessStore (hash loc (ShowL_nat i)) mm of None ⇒ False
            | Some (MValue val) ⇒ (case MTValue t of MTArray n mtypes ⇒ False
                                    | MTValue typ ⇒ MCon (MTValue t) mm (hash loc (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTValue t of MTArray len' arr' ⇒ MCon (MTValue t) mm loc2
                                      | MTValue Types ⇒ False)"
    using assms(1) MCon.simps(2)[of len "MTValue t" mm loc] by simp
  show "∃val. accessStore (hash loc (ShowL_nat i)) mm = Some (MValue val) "
  proof(cases "accessStore (hash loc (ShowL_nat i)) mm")
    case None
    then show ?thesis using a10 by simp
  next
    case (Some a)
    then show ?thesis
    proof(cases a)
      case (MValue x1)
      then show ?thesis using a10 Some by auto
    next
      case (MPointer x2)
      then show ?thesis using a10 Some by auto
    qed
  qed
qed

lemma CompTypeRemainsMCon:
  assumes "CompMemType mem len arr t2 p' stl1"
    and "MCon (MTArray len arr) mem p'"
  shows "MCon t2 mem stl1" using assms
proof(induction arr arbitrary: len p')
  case (MTArray x1 arr)
  then have a10:"∀i<len.
            (case accessStore (hash p' (ShowL_nat i)) mem of
              Some (MPointer loc2) ⇒ MCon (MTArray x1 arr) mem loc2) ∧
            ((∃p. accessStore p' mem = Some (MPointer p)) ∨ accessStore p' mem = None)"
    using MCon.simps(2)[of len "MTArray x1 arr" mem p']
    by (metis MConArrayPointers mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5))
```

```
  have "(∃i<len. ∃l. accessStore (hash p' (ShowL_nat i)) mem = Some (MPointer l)
        ∧ (l = stl1 ∧ MTArray x1 arr = t2 ∨ CompMemType mem x1 arr t2 l stl1))"
    using MTArray CompMemType.simps(2)[of mem len x1 arr t2 p' stl1] by auto
  then obtain i l where idef:"i<len ∧ accessStore (hash p' (ShowL_nat i)) mem = Some (MPointer l)
        ∧ (l = stl1 ∧ MTArray x1 arr = t2 ∨ CompMemType mem x1 arr t2 l stl1)" by blast
  then show ?case
  proof(cases "l = stl1 ∧ MTArray x1 arr = t2")
    case True
    then show ?thesis using a10 idef by auto
  next
    case False
    then show ?thesis using MTArray.IH[of x1 l] a10 idef by fastforce
  qed
next
  case (MTValue x)
  then have a10:"∀i<len.
            (case accessStore (hash p' (ShowL_nat i)) mem of None ⇒ False
             | Some (MValue val) ⇒ (case MTValue x of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTValue x) mem (hash p' (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTValue x of MTArray len' arr' ⇒ MCon (MTValue x) mem
loc2 | MTValue Types ⇒ False)) ∧
            ((∃p. accessStore p' mem = Some (MPointer p)) ∨ accessStore p' mem = None)"
    using MCon.simps(2)[of len "MTValue x" mem p'] by simp
  have a20:"(t2 = MTValue x ∧ (∃i<len. hash p' (ShowL_nat i) = stl1))" using MTValue
CompMemType.simps(1)[of mem len x t2 p' stl1] by auto
  then obtain i where idef:"i<len ∧ hash p' (ShowL_nat i) = stl1" by auto
  then have "MCon (MTValue x) mem (hash p' (ShowL_nat i))" using a10
    by (metis MCon_sub_MTVal_imps_val MTValue.prems(2) mtypes.simps(6) memoryvalue.simps(5)
Option.option.simps(5))
  then show ?case using idef a20 by auto
qed

lemma CompMemJustType_refl: "CompMemJustType t t"
  by (cases t) simp_all
```

Two sub-types of a common type that are both consistent at the same location are equal.

```
lemma MCon_CompMemJustType_unique:
  assumes "CompMemJustType t t1" and "CompMemJustType t t2"
    and "MCon t1 mem l" and "MCon t2 mem l"
  shows "t1 = t2"
proof -
  from CompMemJustTypes_trns[OF assms(1,2)]
  consider "CompMemJustType t1 t2" | "CompMemJustType t2 t1" by blast
  then show ?thesis
  proof cases
    case 1
    show ?thesis
    proof (cases t1)
      case (MTValue v)
      then show ?thesis using 1 by simp
    next
      case (MTArray n a)
      then show ?thesis using 1 assms(3,4) MConSubTypes[of n a mem l] by auto
    qed
  next
    case 2
    show ?thesis
    proof (cases t2)
      case (MTValue v)
      then show ?thesis using 2 by simp
    next
      case (MTArray n a)
      then show ?thesis using 2 assms(3,4) MConSubTypes[of n a mem l] by auto
    qed
```

```
  qed
qed
```

5.3 Consistency under memory changes

```
lemma GetAllMemoryLocations_Arry_Contains_Indexs:
  assumes "MCon (MTArray len2 arr2) mem stl2"
  shows "∀i<len2. hash stl2 (ShowL_nat i) ∈ GetAllMemoryLocations (MTArray len2 arr2) stl2 mem" using
assms
proof(cases arr2)
  case (MTArray x1 arr2')
  show ?thesis
  proof intros
    fix i
    assume *:"i<len2"
    then have "∃c. accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer c)"
      using assms MTArray by (metis MConArrayPointers not_gr0 not_less0)
    then show "hash stl2 (ShowL_nat i) ∈ GetAllMemoryLocations (MTArray len2 arr2) stl2 mem"
      unfolding GetAllMemoryLocations.simps using * by fastforce
  qed
next
  case (MTValue x)
  have exp:"GetAllMemoryLocations (MTArray len2 (MTValue x)) stl2 mem = (⋃i∈{0..<len2}.
        case accessStore (hash stl2 (ShowL_nat i)) mem of None ⇒ {} | Some (MValue xa) ⇒ {hash
          stl2 (ShowL_nat i)} ∪ {hash stl2 (ShowL_nat i)} | Some (MPointer ptr)
          ⇒ {hash stl2 (ShowL_nat i)} ∪ {ptr})"
    unfolding GetAllMemoryLocations.simps by blast
  show ?thesis
  proof intros
    fix i assume *:"i<len2"
    then have "i ∈ {0..<len2}" by simp
    then have exp':"(case accessStore (hash stl2 (ShowL_nat i)) mem of None ⇒ {} | Some (MValue xa)
        ⇒ {hash stl2 (ShowL_nat i)} ∪ {hash stl2 (ShowL_nat i)} | Some (MPointer
        ptr) ⇒ {hash stl2 (ShowL_nat i)} ∪ {ptr})
⊆ GetAllMemoryLocations (MTArray len2 (MTValue x)) stl2 mem" using exp by blast
    have "∃c. accessStore (hash stl2 (ShowL_nat i)) mem = Some (MValue c)" using MTValue assms *
      using MCon_sub_MTVal_imps_val by blast
    then have "(case accessStore (hash stl2 (ShowL_nat i)) mem of None ⇒ {} | Some (MValue xa) ⇒
        {hash stl2 (ShowL_nat i)} ∪ {hash stl2 (ShowL_nat i)} | Some (MPointer ptr)
        ⇒ {hash stl2 (ShowL_nat i)} ∪ {ptr})
                  = {hash stl2 (ShowL_nat i)}" by auto
    then have "{hash stl2 (ShowL_nat i)} ⊆ GetAllMemoryLocations (MTArray len2 (MTValue x)) stl2 mem"
using exp' by simp
    then show "hash stl2 (ShowL_nat i) ∈ GetAllMemoryLocations (MTArray len2 arr2) stl2 mem" using
MTValue by blast
  qed
qed

lemma MCon_imps_sub_Mcon:
  assumes "MCon (MTArray arLen t) srcMem srcl"
  shows "∀x::nat<arLen. (∀l. accessStore (hash srcl (ShowL_nat x)) srcMem = Some (MPointer l) ⟶ MCon
t srcMem l)
      ∧ (∀val. accessStore (hash srcl (ShowL_nat x)) srcMem = Some (MValue val) ⟶ MCon t srcMem (hash
srcl (ShowL_nat x)))"
  using assms
proof(induction t arbitrary: arLen srcl)
  case (MTArray x1 t)
  show ?case
  proof(intros)
    fix x l
    assume *:"x<arLen"
      and "accessStore (hash srcl (ShowL_nat x)) srcMem = Some (MPointer l)"
    then show "MCon (MTArray x1 t) srcMem l" using MTArray
      by (meson CompTypeRemainsMCon CompMemType.simps(2))
```

```
  next
    fix x val
    assume *:"x<arLen"
      and "accessStore (hash srcl (ShowL_nat x)) srcMem = Some (MValue val)"
    then show "MCon (MTArray x1 t) srcMem (hash srcl (ShowL_nat x))" using MTArray by auto
  qed
next
  case (MTValue x)
  show ?case
  proof(intros)
    fix xa l
    assume *:"xa<arLen"
      and **:"accessStore (hash srcl (ShowL_nat xa)) srcMem = Some (MPointer l)"
    show "MCon (MTValue x) srcMem l" using * **
      using MTValue by force
  next
    fix xa val
    assume *:"xa<arLen"
      and **:"accessStore (hash srcl (ShowL_nat xa)) srcMem = Some (MValue val)"
    show "MCon (MTValue x) srcMem (hash srcl (ShowL_nat xa))" using * ** MTValue by auto
  qed
qed

lemma MconSameTypeSameAccessWithTyping:
  assumes "MCon (MTArray x t) m1 l1"
    and "MCon (MTArray x t) m2 l2"
  shows "∀i1<x. ∀i2<x. (∀l. accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MPointer l)
      ⟶ (∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l') ∧ MCon t m1 l
      ∧ MCon t m2 l'))
        ∧ (∀val . accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MValue val) ⟶
            (∃val'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MValue val')∧ MCon t m1 (hash l1
            (ShowL_nat i1)) ∧ MCon t m2 (hash l2 (ShowL_nat i2))))"
  using assms
proof(induction t arbitrary:x l1 l2)
  case (MTArray x1 t)
  show ?case
  proof(intros)
    fix i1 i2 l
    assume *:"i1<x" and ***:"i2<x"
      and **:"accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MPointer l)"
    then have "MCon (MTArray x1 t) m1 l" using MTArray
      by (meson CompTypeRemainsMCon CompMemType.simps(2))
    moreover have "∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l')" using MTArray
*** ** "*" MConArrayPointers gr_zeroI less_nat_zero_code by metis
    ultimately show "∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l') ∧ MCon (MTArray
x1 t) m1 l ∧ MCon (MTArray x1 t) m2 l'"
      using MTArray.IH ** ***
      by (meson "*" CompTypeRemainsMCon MTArray.prems(2) CompMemType.simps(2))

  next
    fix i1 i2 val
    assume *:"i1<x" and ***:"i2<x"
      and **:" accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MValue val)"
    show "∃val'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MValue val') ∧ MCon (MTArray x1 t) m1
       (hash l1 (ShowL_nat i1)) ∧ MCon (MTArray x1 t) m2 (hash l2 (ShowL_nat i2))"
      using "*" "**" MTArray.prems(1) by fastforce
  qed
next
  case (MTValue x')
  show ?case
  proof(intros)
    fix i1 i2 l
    assume *:"i1<x" and ***:"i2<x"
      and **:"accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MPointer l)"
```

```
    show "∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l') ∧ MCon (MTValue x') m1 l ∧
MCon (MTValue x') m2 l'" using * ** MTValue by auto
  next
    fix i1 i2 val
    assume *:"i1<x" and ****:"i2<x"
      and **:"accessStore (hash l1 (ShowL_nat i1)) m1 = Some (MValue val)"
    have ***:"(case accessStore (hash l2 (ShowL_nat i2)) m2 of None ⇒ False
                | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTValue x') m2 (hash l2 (ShowL_nat i2)))
                | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') m2
loc2 | MTValue Types ⇒ False))"
      using MTValue(2) MCon.simps(2)[of x "MTValue x'" m2 l2] * **** by simp
    then have "∃l. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MValue l)"
      by (auto split:option.splits mtypes.splits memoryvalue.splits)
    moreover have "MCon (MTValue x') m1 (hash l1 (ShowL_nat i1))" using * ** MTValue by auto
    ultimately show "∃val'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MValue val') ∧ MCon
(MTValue x') m1 (hash l1 (ShowL_nat i1))
                    ∧ MCon (MTValue x') m2 (hash l2 (ShowL_nat i2))" using *** by auto
  qed
qed


lemma SubPtrs_top:
  assumes "LSubPrefL2 stl2 (ShowL_nat (Toploc mem))"
    and "∀l l'. LSubPrefL2 l (ShowL_nat (Toploc mem)) ∧ accessStore l mem' = Some (MPointer l') ⟶ l'
= l"
    and "TypedMemSubPrefPtrs mem' x11 x12 stl2 dloc1"
  shows "LSubPrefL2 dloc1 (ShowL_nat (Toploc mem))"
proof(cases x12)
  case (MTArray x11' x12')

  then obtain i l where idef:"i<x11 ∧ accessStore (hash stl2 (ShowL_nat i)) mem' = Some (MPointer l) ∧
(l = dloc1 ∨ TypedMemSubPrefPtrs mem' x11' x12' l dloc1)"
    using assms by auto
  then have lsubloc:"LSubPrefL2 (hash stl2 (ShowL_nat i)) (ShowL_nat (Toploc mem))" using LSubPrefL2_def
    using assms(1) hash_suffixes_associative by auto
  then have lSelfPoint:"l = (hash stl2 (ShowL_nat i))" using assms idef by simp

  have lsubloc:"LSubPrefL2 l (ShowL_nat (Toploc mem))" using lsubloc lSelfPoint by simp

  then show ?thesis
  proof(cases "l = dloc1")
    case True
    then show ?thesis
      using lsubloc lSelfPoint by auto
  next
    case False
    then have lneq2:"∀i<x11'. LSubPrefL2 (hash l (ShowL_nat i)) (ShowL_nat (Toploc mem))" using
lSelfPoint idef lsubloc
      by (metis LSubPrefL2_def Not_Sub_More_Specific)
    have "TypedMemSubPrefPtrs mem' x11' x12' l dloc1" using idef False by simp
    then show ?thesis using lneq2 lsubloc
    proof(induction x12' arbitrary:x11' l)
      case (MTArray x1 x12')
      have "∃i'<x11'. ∃l'. accessStore (hash l (ShowL_nat i')) mem' = Some (MPointer l') ∧ (l' = dloc1
∨ TypedMemSubPrefPtrs mem' x1 x12' l' dloc1)"
        using MTArray(2) unfolding TypedMemSubPrefPtrs.simps by simp
      then obtain i' l' where i'def:"i'<x11' ∧ accessStore (hash l (ShowL_nat i')) mem' = Some
(MPointer l') ∧
          (l' = dloc1 ∨ TypedMemSubPrefPtrs mem' x1 x12' l' dloc1)" by blast

      have l'subloc:"LSubPrefL2 (hash l (ShowL_nat i')) (ShowL_nat (Toploc mem))" using i'def
LSubPrefL2_def
        using MTArray.prems(2) by auto
```

```
      then have l'SP:"(hash l (ShowL_nat i')) = l'" using i'def assms by auto
      show ?case
      proof(cases "l' = dloc1")
        case True
        then show ?thesis using l'SP l'subloc by simp
      next
        case False
        then have "TypedMemSubPrefPtrs mem' x1 x12' l' dloc1" using i'def by simp
        moreover have "∀i<x1. LSubPrefL2 (hash l' (ShowL_nat i)) (ShowL_nat(Toploc mem))"
          using l'subloc MemLSubPrefL2_specific_imps_general l'SP l'subloc
          by (metis LSubPrefL2_def Not_Sub_More_Specific)
        ultimately show ?thesis using MTArray.IH[of x1 l'] l'subloc l'SP by blast
      qed
    next
      case (MTValue x)


      then show ?case by auto
    qed
  qed
next
  case (MTValue x2)
  then show ?thesis
    using LSubPrefL2_def hash_suffixes_associative
    using assms(1,3) by auto
qed


lemma TypedMemSubPrefPtrs_trns:
  assumes "TypedMemSubPrefPtrs mem x2 t2 l2 l3"
    and "MCon (MTArray x1 t1) mem l1"
    and "CompMemType mem x1 t1 (MTArray x2 t2) l1 l2"
  shows "TypedMemSubPrefPtrs mem x1 t1 l1 l3" using assms
proof(induction t1 arbitrary:x1 l1)
  case (MTArray x1' t1')
  then have "(∃i<x1. ∃l. accessStore (hash l1 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l = l2
      ∧ MTArray x1' t1' = MTArray x2 t2 ∨ CompMemType mem x1' t1' (MTArray x2 t2) l l2))"
    using CompMemType.simps(2)[of mem x1 x1' t1' "MTArray x2 t2" l1 l2] by auto
  then obtain i l where idef:"i<x1 ∧ accessStore (hash l1 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l
= l2
      ∧ MTArray x1' t1' = MTArray x2 t2 ∨ CompMemType mem x1' t1' (MTArray x2 t2) l l2)"
    by blast
  then have mcond:"MCon (MTArray x1' t1') mem l" using MTArray(3)
    using CompTypeRemainsMCon CompMemType.simps(2) by blast

  then show ?case using TypedMemSubPrefPtrs.simps(2)[of mem x1 x1' t1' l1 l3] idef
    using MTArray.IH assms(1) assms(2) by blast
next
  case (MTValue x)
  then show ?case by simp
qed

lemma allocateMCon:
  assumes "MCon t mm loc"
    and "snd (allocate mm) = m'"
  shows "MCon t m' loc"
proof -
  have "∀locs. accessStore locs mm = accessStore locs m'"
    using assms(2) unfolding allocate_def accessStore_def by auto
  then show ?thesis using assms(1)
  proof(induction t arbitrary:loc)
    case (MTArray x1 t)

    have "∀i<x1.
```

```
            (case accessStore (hash loc (ShowL_nat i)) m' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTArray l a ⇒ False | MTValue typ ⇒ MCon t m' (hash
loc (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t m' loc2 | MTValue val ⇒
False))"
    proof intros
      fix i
      assume *:"i<x1"
      then show "(case accessStore (hash loc (ShowL_nat i)) m' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTArray l a ⇒ False | MTValue typ ⇒ MCon t m' (hash
loc (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t m' loc2 | MTValue val ⇒
False))"
      proof(cases t)
        case mta:(MTArray x11 x12)
        then obtain v where vDef:"accessStore (hash loc (ShowL_nat i)) mm = Some (MPointer v)
                                  ∧ MCon t mm v" using *
          by (metis MConArrayPointers MCon_imps_sub_Mcon MTArray.prems(2) MCon.simps(2))
        then show ?thesis using * MTArray mta by fastforce
      next
        case (MTValue x2)
        then obtain v where vDef:"accessStore (hash loc (ShowL_nat i)) mm = Some (MValue v)
                                  ∧ MCon t mm (hash loc (ShowL_nat i))" using *
          using MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val MTArray.prems(2) by presburger
        then show ?thesis using MTArray.prems(1) MTValue by auto
      qed
    qed
    then show ?case unfolding MCon.simps using MTArray.prems
      by (metis MCon.simps(2))
  next
    case (MTValue x)
    then show ?case by simp
  qed
qed


lemma compMemTypes_trns:
  assumes "CompMemType mem len' arr' tp2 p' stl1"
    and "CompMemType mem len arr (MTArray len' arr') p'' p'"
  shows "CompMemType mem len arr tp2 p'' stl1" using assms
proof(induction arr arbitrary:p'' len)
  case (MTArray x1 arr)
  then show ?case by fastforce
next
  case (MTValue x)
  then show ?case by auto
qed

lemma Not_Sub_More_Specific_more_speific:
  assumes "¬ TypedMemSubPref destl' destl (MTArray l t)"
  shows "¬ (x<l ∧ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t)" using assms by auto


lemma limitedMemoryChange:
  assumes "¬TypedMemSubPrefPtrs mem x t p'' prnt"
    and "p'' ≠ prnt"
    and "∀locs. locs ≠ (hash prnt (ShowL_nat i)) ⟶ accessStore locs mem = accessStore locs mem'"
    and "∃p. accessStore (hash prnt (ShowL_nat i)) mem' = Some (MPointer p)"
    and "MCon (MTArray x t) mem p'''
  shows "MCon (MTArray x t) mem' p'''" using assms
proof (induction t arbitrary: x p'')
  case (MTArray x1 t)
  then have "x ≠ 0" using MCon.simps by blast
  moreover have "(∃p. accessStore p'' mem' = Some (MPointer p)) ∨ accessStore p'' mem' = None"
```

```
  proof(cases "p'' = (hash prnt (ShowL_nat i))")
    case True
    then show ?thesis using MTArray by blast
  next
    case False
    then have "accessStore p'' mem = accessStore p'' mem'" using MTArray by simp
    then show ?thesis using MTArray MCon.simps ‹x≠0› by simp
  qed
  moreover have "∀i<x. (case accessStore (hash p'' (ShowL_nat i)) mem' of None ⇒ False
                | Some (MValue val) ⇒ (case (MTArray x1 t) of MTArray l a ⇒ False | MTValue typ ⇒
MCon
                    (MTArray x1 t) mem' (hash p'' (ShowL_nat i)))
                | Some (MPointer loc2) ⇒ (case (MTArray x1 t) of MTArray len' arr' ⇒ MCon (MTArray x1
t) mem' loc2 | MTValue Types ⇒ False))"
  proof intros
    fix ii assume iiDef:"ii<x"
    show " case accessStore (hash p'' (ShowL_nat ii)) mem' of None ⇒ False
        | Some (MValue val) ⇒ (case (MTArray x1 t) of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTArray x1 t) mem' (hash p'' (ShowL_nat ii)))
        | Some (MPointer loc2) ⇒ (case (MTArray x1 t) of MTArray len' arr' ⇒ MCon (MTArray x1 t) mem'
loc2 | MTValue Types ⇒ False)"
    proof(cases " (hash p'' (ShowL_nat ii)) = (hash prnt (ShowL_nat i))")
      case True
      then have "p'' = prnt" using ShowLNatDot
        using hash_injective by blast
      then show ?thesis using MTArray by simp
    next
      case False
      then have same:"accessStore (hash p'' (ShowL_nat ii)) mem = accessStore (hash p'' (ShowL_nat ii))
mem'" using MTArray by simp
      then obtain v where "accessStore (hash p'' (ShowL_nat ii)) mem' = Some (MPointer v) ∧ MCon
(MTArray x1 t) mem v"
        using MTArray iiDef MCon_imps_sub_Mcon
        by (metis MConArrayPointers bot_nat_0.not_eq_extremum calculation(1))
      then show ?thesis using MTArray.IH[of x1 v ] MTArray.prems iiDef same by fastforce
    qed
  qed
  ultimately show ?case by simp
next
  case (MTValue x')
  then have "x ≠ 0" using MCon.simps by auto
  moreover have "(∃p. accessStore p'' mem' = Some (MPointer p)) ∨ accessStore p'' mem' = None"
  proof(cases "p'' = (hash prnt (ShowL_nat i))")
    case True
    then show ?thesis using MTValue by blast
  next
    case False
    then have "accessStore p'' mem = accessStore p'' mem'" using MTValue by simp
    then show ?thesis using MTValue MCon.simps ‹x≠0› by simp
  qed
  moreover have "∀i<x. (case accessStore (hash p'' (ShowL_nat i)) mem' of None ⇒ False
                | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTValue x') mem' (hash p'' (ShowL_nat i)))
                | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x')
mem' loc2 | MTValue Types ⇒ False))"
  proof intros
    fix ii assume iiDef:"ii<x"
    show " case accessStore (hash p'' (ShowL_nat ii)) mem' of None ⇒ False
        | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTValue
x') mem' (hash p'' (ShowL_nat ii)))
        | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') mem' loc2
| MTValue Types ⇒ False)"
    proof(cases " (hash p'' (ShowL_nat ii)) = (hash prnt (ShowL_nat i))")
      case True
```

```
      then have "p'' = prnt" using ShowLNatDot
        using hash_injective by blast
      then show ?thesis using MTValue by simp
    next
      case False
      then have same:"accessStore (hash p'' (ShowL_nat ii)) mem = accessStore (hash p'' (ShowL_nat ii))
mem'" using MTValue by simp
      then obtain v where "accessStore (hash p'' (ShowL_nat ii)) mem' = Some (MValue v) ∧ MCon (MTValue
x') mem (hash p'' (ShowL_nat ii))" using MTValue
          MCon_sub_MTVal_imps_val iiDef by (metis MCon_imps_sub_Mcon)
      then show ?thesis using same by auto
    qed
  qed
  ultimately show ?case using MCon.simps(2)[of x "MTValue x'" mem p''] by simp
qed

lemma mconCopySingle:
  assumes "∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cd' = accessStore locs c''"
    and"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc cd' = None"
    and "MCon struct cd' x2"
  shows "MCon struct c'' x2" using assms(3)
proof(induction struct arbitrary:x2)
  case (MTArray x1 struct')
  then show ?case
  proof(cases "x1 = 0")
    case True
    then show ?thesis using MTArray by simp
  next
    case False
    then have a10:"∀ (i::nat)<x1.
            case accessStore (hash x2 (ShowL_nat i)) cd' of None ⇒ False
            | Some (MValue val) ⇒ (case struct' of MTValue typ ⇒ MCon struct' cd' (hash x2 (ShowL_nat
i)) | _ ⇒ False)
            | Some (MPointer loc2) ⇒ (case struct' of MTArray len' arr' ⇒ MCon struct' cd' loc2 |
MTValue Types ⇒ False)"
      using MCon.simps(2)[of x1 struct' cd' x2] MTArray by simp
    have "∀ i<x1.
            case accessStore (hash x2 (ShowL_nat i)) c'' of None ⇒ False
            | Some (MValue val) ⇒ (case struct' of MTValue typ ⇒ MCon struct' c'' (hash x2 (ShowL_nat
i)) | _ ⇒ False)
            | Some (MPointer loc2) ⇒ (case struct' of MTArray len' arr' ⇒ MCon struct' c'' loc2 |
MTValue Types ⇒ False)"
    proof intros
      fix i assume a20:"i<x1"
      then obtain x where xdef:"accessStore (hash x2 (ShowL_nat i)) cd' = Some x" using a10 by
fastforce
      then have "¬ LSubPrefL2 (hash x2 (ShowL_nat i)) l" using assms by auto
      then have same:"accessStore (hash x2 (ShowL_nat i)) c'' = Some x" using assms(1) xdef by simp
      then show " case accessStore (hash x2 (ShowL_nat i)) c'' of None ⇒ False
            | Some (MValue val) ⇒ (case struct' of MTValue typ ⇒ MCon struct' c'' (hash x2 (ShowL_nat
i)) | _ ⇒ False)
            | Some (MPointer loc2) ⇒ (case struct' of MTArray len' arr' ⇒ MCon struct' c'' loc2 |
MTValue Types ⇒ False)"
      proof(cases "x")
        case (MValue x1')
        then show ?thesis
        proof(cases struct')
          case (MTArray x11 x12)
          then show ?thesis using a10 a20 xdef MValue by auto
        next
          case (MTValue x2')
          then have "MCon struct' cd' (hash x2 (ShowL_nat i))" using a10 a20 xdef MValue by auto
          then show ?thesis using same MValue MTValue MTArray(1) by force
        qed
```

```
      next
        case (MPointer x2)
        then show ?thesis
        proof(cases struct')
          case mtr:(MTArray x11 x12)
          then have "MCon struct' cd' x2" using a10 a20 xdef MPointer
            by fastforce
          then show ?thesis using same MPointer MTArray(1) mtr
            by fastforce
        next
          case (MTValue x2')
          then show ?thesis using a10 a20 xdef MPointer by auto
        qed
      qed

    qed
    moreover have "(∃p. accessStore x2 c'' = Some (MPointer p)) ∨ accessStore x2 c'' = None"
      by (metis (no_types, lifting) MTArray.prems LSubPrefL2_def assms(1) assms(2)
hash_suffixes_associative option.case_eq_if MCon.simps(2))
    ultimately show ?thesis using MCon.simps(2)[of x1 struct' c'' x2]
      using False by auto
  qed
next
  case (MTValue x)
  then have "(case accessStore x2 cd' of None ⇒ False
            | Some (MValue t) ⇒ typeCon x t
            | Some (MPointer t) ⇒ False)"
    using MCon.simps(1)[of x cd' x2] by simp
  then obtain t where "accessStore x2 cd' = Some (MValue t)"
    by (simp split:option.splits memoryvalue.splits)
  then have "¬LSubPrefL2 x2 l" using assms(2) by auto
  then show ?case using MTValue assms by simp
qed


lemma mcon_accessStore:
  assumes "MCon (MTArray len arr) cd loc"
    and "len > 0"
    and "i<len"
  obtains (1) loc2 len' arr'
  where "accessStore (hash loc (ShowL_nat i)) cd = Some (MPointer loc2)"
    and "arr = MTArray len' arr'"
  | (2) val tp
  where "accessStore (hash loc (ShowL_nat i)) cd = Some (MValue val)"
    and "arr = MTValue tp"
proof(cases "accessStore (hash loc (ShowL_nat i)) cd")
  case None
  then show ?thesis using assms MCon.simps by auto
next
  case (Some a)
  then show ?thesis
  proof(cases a)
    case (MValue x1)
    then show ?thesis
    proof(cases arr)
      case (MTArray x11 x12)
      then show ?thesis using Some MValue MCon.simps assms by auto
    next
      case (MTValue x2)
      then show ?thesis using Some MValue MCon.simps assms by (simp add: "2")
    qed
  next
    case (MPointer x2)
    then show ?thesis
```

```
    proof(cases arr)
      case (MTArray x11 x12)
      then show ?thesis using Some MPointer MCon.simps assms by (simp add: "1")
    next
      case (MTValue x2)
      then show ?thesis using Some MPointer MCon.simps assms by auto
    qed
  qed
qed

lemma SameMCon_imps_MConAccessSame:
  assumes "MCon t m1 l1"
    and "MCon t m2 l2"
  shows "MConAccessSame l1 l2 t m1 m2" using assms
proof(induction t arbitrary: l1 l2)
  case (MTArray x1 t)
  show ?case unfolding MConAccessSame.simps
  proof(intros)
    fix i1 i2 l
    assume *:"i1 <x1" and **:"i2<x1" and ***:" accessStore (hash l1 (ShowL_nat i1)) m1 = Some
(MPointer l)"
    then have "∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l')" using
MTArray.prems(2)
      by (metis MTArray.prems(1) mtypes.distinct(1) memoryvalue.distinct(1) gr_zeroI less_nat_zero_code
mcon_accessStore option.inject)
    then obtain l' where l'def:"accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l')" by blast
    moreover have "MCon t m1 l" using MTArray.prems(1) * ***
      using MCon_imps_sub_Mcon by blast
    moreover have "MCon t m2 l'" using ** l'def MTArray.prems(2)
      using MCon_imps_sub_Mcon by blast
    ultimately have "MConAccessSame l l' t m1 m2" using MTArray.IH[of l l'] using MTArray.prems *** *
** by simp
    then show "∃l'. accessStore (hash l2 (ShowL_nat i2)) m2 = Some (MPointer l') ∧ MConAccessSame l l'
t m1 m2 " using l'def by simp
  qed
next
  case (MTValue x)
  then have mc1:" case accessStore l1 m1 of None ⇒ False | Some (MValue xb) ⇒ typeCon x xb | Some
(MPointer t) ⇒ False"
    unfolding MCon.simps by simp
  have mc2:"case accessStore l2 m2 of None ⇒ False | Some (MValue xb) ⇒ typeCon x xb | Some (MPointer
t) ⇒ False"
    using MTValue unfolding MCon.simps by simp
  have "∃val. accessStore l1 m1 = Some (MValue val) ∧ (∃val'. accessStore l2 m2 = Some (MValue val'))"
    using mc2 mc1 by (auto split:option.splits memoryvalue.splits)
  then show ?case unfolding MConAccessSame.simps by blast
qed

lemma MCon_imps_TypedMemSubPref_Some:
  assumes "TypedMemSubPref x' loc t"
    and "MCon t m loc"
    and "∀l l'. TypedMemSubPref l loc t ∧ accessStore l m = Some (MPointer l') ⟶ l' = l"
  shows "∃v. accessStore x' m = Some v"
  using assms
proof(induction t arbitrary:loc)
  case (MTArray x1 t)
  then have a1:"(∃i<x1. TypedMemSubPref x' (hash loc (ShowL_nat i)) t ∨ x' = hash loc (ShowL_nat i))"
    using TypedMemSubPref.simps(2)[of x' loc x1 t] by simp
  then have "x1 > 0" using not_less_zero by auto
  then have mcon_array: "x1 > 0 ∧ (∀j<x1. case accessStore (hash loc (ShowL_nat j)) m of
                                    None ⇒ False
                                  | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t m (hash loc
(ShowL_nat j)) | _ ⇒ False)
                                  | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t m
```

```
loc2 | MTValue typ ⇒ False))
                        ∧ (∃p. accessStore loc m = Some (MPointer p) ∨ accessStore loc m = None)"
    using MTArray.prems(2) MCon.simps(2)[of x1 t m loc] by simp
  then have som:"∀j<x1. ∃v. accessStore (hash loc (ShowL_nat j)) m = Some v"
    using mcon_accessStore
    by fastforce
  then show ?case
  proof(cases "∃i<x1. x' = hash loc (ShowL_nat i)")
    case True
    then show ?thesis using som by blast
  next
    case False
    then obtain i where iDef:"i<x1 ∧ TypedMemSubPref x' (hash loc (ShowL_nat i)) t" using a1 by blast
    then obtain v where vDef: "accessStore (hash loc (ShowL_nat i)) m = Some v" using som by blast
    then show ?thesis
    proof(cases v)
      case (MValue x1)
      then show ?thesis using iDef MTArray.IH vDef mcon_array
        by (metis MTArray.prems(2) memoryvalue.simps(4) TypedMemSubPref.simps(1) mcon_accessStore
option.inject)
    next
      case (MPointer x2)
      then have "x2 = (hash loc (ShowL_nat i))" using iDef MTArray.prems(3)
        using TypedMemSubPref.simps(2) vDef by blast
      then show ?thesis using MTArray.IH iDef vDef mcon_array
        by (metis MCon_imps_sub_Mcon MPointer MTArray.prems(2,3) TypedMemSubPref.simps(2))
    qed
  qed
next
  case (MTValue x)
  then show ?case
    by (metis MConAccessSame.simps(1) SameMCon_imps_MConAccessSame TypedMemSubPref.simps(1))
qed

lemma mcon_typedptrs_imps_existence:
  assumes "MCon (MTArray x t) m1 l1"
  shows "∀l1'. TypedMemSubPrefPtrs m1 x t l1 l1' ⟶ (∃t. MCon t m1 l1')" using assms(1)
proof(induction t arbitrary:x l1)
  case (MTArray x1 t)
  then show ?case
    by (metis MCon_imps_sub_Mcon TypedMemSubPrefPtrs.simps(2))
next
  case (MTValue x)
  then show ?case
    using CompTypeRemainsMCon TypedMemSubPrefPtrs.simps(1) CompMemType.simps(1) by blast
qed

lemma memSet_selfPoint:
  assumes "MCon t' m tL"
    and "x'' ∈ GetAllMemoryLocations t' tL m"
    and "∀l l'. TypedMemSubPref l tL t' ∧ accessStore l m = Some (MPointer l') ⟶ l' = l"
    and "accessStore x'' m = Some (MPointer locs')"
  shows "locs' = x''" using assms
proof(induction t' arbitrary:tL)
  case (MTArray x1 t')
  obtain i where idef: "i∈{0..<x1} ∧ x'' ∈ (case accessStore (hash tL (ShowL_nat i)) m of None
      ⇒ {} | Some (MValue xa) ⇒ {hash tL (ShowL_nat i)} ∪ GetAllMemoryLocations t' (hash
      tL (ShowL_nat i)) m
          | Some (MPointer ptr) ⇒ {hash tL (ShowL_nat i)} ∪ GetAllMemoryLocations t' ptr m)"
    using MTArray.prems unfolding GetAllMemoryLocations.simps by blast
  then consider (ptr) ptr where "i<x1 ∧ accessStore (hash tL (ShowL_nat i)) m = Some (MPointer ptr) ∧
x''
      ∈ {hash tL (ShowL_nat i)} ∪ GetAllMemoryLocations t' ptr m"
    | (val) val where "i<x1 ∧ accessStore (hash tL (ShowL_nat i)) m = Some (MValue val) ∧ x'' ∈ {hash
```

```
tL
        (ShowL_nat i)} ∪ GetAllMemoryLocations t' (hash tL (ShowL_nat i)) m"
    by (auto split:option.splits memoryvalue.splits)
  then show ?case
  proof(cases)
    case ptr
    then have subL:"TypedMemSubPref (hash tL (ShowL_nat i)) tL (MTArray x1 t')" by auto
    then have ptrSame:"ptr = (hash tL (ShowL_nat i))" using MTArray.prems(3) ptr by blast

    then show ?thesis
    proof(cases "x'' = hash tL (ShowL_nat i)")
      case True
      then show ?thesis
        using MTArray.prems(3) assms(4) subL by blast
    next
      case False
      then have a0:"x'' ∈ GetAllMemoryLocations t' ptr m" using ptr by simp
      moreover have "MCon t' m ptr" using MTArray.prems(1) ptr
        using MCon_imps_sub_Mcon by blast
      moreover have "∀l l'. TypedMemSubPref l ptr t' ∧ accessStore l m = Some (MPointer l') ⟶ l' =
l"
        using MTArray.prems(3) Not_Sub_More_Specific_more_speific ptr ptrSame by blast
      ultimately have "locs' = x''" using MTArray.IH[OF _ a0] MTArray.prems(4) by blast
      then show ?thesis by blast
    qed
  next
    case val
    then have "∃v'. t' = MTValue v'" using MTArray.prems(1)
      by (metis memoryvalue.distinct(1) mcon_accessStore neq0_conv option.inject order_less_trans)
    then have "x'' = hash tL (ShowL_nat i)" using val by auto
    then show ?thesis using MTArray val by auto
  qed
next
  case (MTValue x)
  then show ?case by auto
qed


lemma moreSpecificTypedSubPref:
  assumes "( ∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer l') ⟶
l' = l)"
  shows "∀i<x. ( ∀l l'. TypedMemSubPref l (hash destl (ShowL_nat i)) t ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l)" using assms(1)
proof (induction t arbitrary:x)
  case (MTArray x1 t)
  show ?case
  proof intros
    fix i l l'
    assume *:"i<x"
      and **:"TypedMemSubPref l (hash destl (ShowL_nat i)) (MTArray x1 t) ∧ accessStore l v' = Some
(MPointer l')"
    then show "l' =l" using MTArray(1) * **
      by (meson MTArray.prems Not_Sub_More_Specific_more_speific)
  qed
next
  case (MTValue x)
  then show ?case by auto
qed

lemma MCon_mem_preserved_disjoint_update:
  assumes "(¬LSubPrefL2 destl' destl) ∧ ¬LSubPrefL2 destl destl' "
  shows "(∀l l'. TypedMemSubPref l destl' t' ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l)
         ∧ (∀destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl' v'')
         ∧ MCon t' v' destl' ⟶ MCon t' v'' destl'" using assms
```

```
proof(induction t' arbitrary:destl' )
  case (MTArray x1 t)
  show ?case
  proof intros
    assume "(∀ l l'. TypedMemSubPref l destl' (MTArray x1 t) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)
              ∧ (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl'
v'')
              ∧ MCon (MTArray x1 t) v' destl'
              "
    then have *:"(∀ l l'. TypedMemSubPref l destl' (MTArray x1 t) ∧ accessStore l v' = Some (MPointer
l') ⟶ l' = l)"
      and **: "MCon (MTArray x1 t) v' destl'"
      and ***:"(∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl'
v'')"
      by blast+


    then show "MCon (MTArray x1 t) v'' destl'"
    proof (cases "x1>0")
      case True
      have a10:"∀ i<x1.
            case accessStore (hash destl' (ShowL_nat i)) v' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v' (hash destl' (ShowL_nat i)) | _
⇒ False)
            | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 | MTValue Types ⇒
False)"
        using ** by auto


      have "∀ i<x1.
            case accessStore (hash destl' (ShowL_nat i)) v'' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' (hash destl' (ShowL_nat i)) | _
⇒ False)
            | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue Types
⇒ False)"
      proof intros
        fix i
        assume iLess: "i<x1"
        show "case accessStore (hash destl' (ShowL_nat i)) v'' of None ⇒ False
                | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' (hash destl' (ShowL_nat i))
| _⇒ False)
                | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue
Types ⇒ False)"
        proof(cases t)
          case mtr:(MTArray x11 x12)
          have b100:"¬ LSubPrefL2 (hash destl' (ShowL_nat i)) destl" using Mutual_NonSub_SpecificNonSub
MTArray by auto
          then have "¬ LSubPrefL2 destl (hash destl' (ShowL_nat i))" using Not_Sub_More_Specific
MTArray by auto
          then have a50:"(∀ l l'. TypedMemSubPref l (hash destl' (ShowL_nat i)) t ∧ accessStore l v' =
Some (MPointer l') ⟶ l' = l)
                         ∧ (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' =
accessStore destl' v'')
                         ∧ MCon t v' (hash destl' (ShowL_nat i)) ⟶
                         MCon t v'' (hash destl' (ShowL_nat i))
            " using MTArray(1)[of "(hash destl' (ShowL_nat i))"] b100 hash_suffixes_associative by blast
          have a55:"∀ l l'. TypedMemSubPref l (hash destl' (ShowL_nat i)) t ∧ accessStore l v' = Some
              (MPointer l') ⟶ l' = l" using * moreSpecificTypedSubPref[of destl' x1 t v'] iLess by
blast
          then have a60:"accessStore (hash destl' (ShowL_nat i)) v' = accessStore (hash destl' (ShowL_nat
i)) v''" using b100 *** by blast
          then show ?thesis
          proof(cases "accessStore (hash destl' (ShowL_nat i)) v'")
```

```
          case None
          then show ?thesis using a10 iLess by auto
        next
          case (Some a)
          then show ?thesis
          proof(cases a)
            case (MValue x1)
            then show ?thesis using a10 iLess mtr Some by auto
          next
            case (MPointer x2')
            then have x2'def:"x2' = (hash destl' (ShowL_nat i))" using a55 Some *
              using iLess TypedMemSubPref.simps(2) by blast
            then have "MCon t v' (x2')" using a10 iLess Some mtr MPointer
              by fastforce
            then have "MCon t v'' x2'" using a50 b100 a55 *** x2'def by auto
            then show ?thesis using Some MPointer mtr a60 by auto
          qed
        qed
      next
        case (MTValue x2)
        have b100:"¬ LSubPrefL2 (hash destl' (ShowL_nat i)) destl" using Mutual_NonSub_SpecificNonSub
MTArray by auto
        then have "¬ LSubPrefL2 destl (hash destl' (ShowL_nat i))" using Not_Sub_More_Specific
MTArray by auto
        then have a50:"(∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore
destl' v'') ∧
                        MCon t v' (hash destl' (ShowL_nat i)) ⟶
                       MCon t v'' (hash destl' (ShowL_nat i))
          " using MTArray(1)[of "(hash destl' (ShowL_nat i))"] b100 using MTValue by force
        then have a60:"accessStore (hash destl' (ShowL_nat i)) v' = accessStore (hash destl' (ShowL_nat
i)) v''" using b100 *** by simp
        then show ?thesis
        proof(cases "accessStore (hash destl' (ShowL_nat i)) v'")
          case None
          then show ?thesis using a10 iLess by auto
        next
          case (Some a)
          then show ?thesis
          proof(cases a)
            case (MValue x1)
            then have "MCon t v' (hash destl' (ShowL_nat i))" using a10 iLess Some MTValue by auto
            then have "MCon t v'' (hash destl' (ShowL_nat i))" using a50 b100 a60 *** Some MValue by
auto
            then show ?thesis using Some MValue MTValue a60 by auto
          next
            case (MPointer x2')
            then show ?thesis using a10 iLess MTValue Some by auto
          qed

        qed

      qed

    qed

    moreover have "(∃p. accessStore destl' v'' = Some (MPointer p)) ∨ accessStore destl' v'' =
None" using ** MCon.simps(2)
      using MTArray.prems True <(∀ l l'. TypedMemSubPref l destl' (MTArray x1 t) ∧ accessStore l v' =
Some (MPointer l')
          ⟶ l' = l) ∧ (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' =
          accessStore destl' v'') ∧ MCon (MTArray x1 t) v' destl'> by auto
    ultimately show ?thesis using MCon.simps(2)[of x1 t v'' destl']
      by (simp add: True)
  next
```

```
      case False
      then show ?thesis using ** by simp
    qed

  qed
next
  case (MTValue x)
  show ?case
  proof intros
    assume "(∀ l l'. TypedMemSubPref l destl' (MTValue x) ∧ accessStore l v' = Some (MPointer l') ⟶
l' = l) ∧
    (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl' v'') ∧
MCon (MTValue x) v' destl' "
    then have *:"(∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl'
v'')"
      and **:"MCon (MTValue x) v' destl'"
      and ***:"(∀ l len tp.
        MTValue x = MTArray len tp ∧ MCon (MTValue x) v' destl' ⟶
        (∀ i<len. accessStore (hash destl' (ShowL_nat i)) v' = Some (MPointer l) ∧ l = hash destl'
(ShowL_nat i)))"
      by auto+
    then have "accessStore destl' v' = accessStore destl' v''" using MTValue by simp
    then show "MCon (MTValue x) v'' destl'" using ** by simp
  qed
qed
```

5.4 Typed sub-prefixes

```
lemma mtype_size_strict_decrease_array: "mtype_size arr < mtype_size (MTArray len arr)"
  by (simp)

lemma neg_MemLSubPrefL2_imps_TypedMemSubPref:
  assumes "¬LSubPrefL2 l1 l2"
  shows "¬TypedMemSubPref l1 l2 t" using assms
proof(induction t arbitrary:l2)
  case (MTArray x1 t)
  then show ?case
    by (simp add: LSubPrefL2_def hash_suffixes_associative)
next
  case (MTValue x)
  then show ?case
    by (simp add: LSubPrefL2_def)
qed

abbreviation mymemory::memoryT
  where "mymemory ≡
    (|Mapping = fmap_of_list
      [(STR ''1.1.0'', MValue STR ''False''),
       (STR ''0.1.0'', MValue STR ''True''),
       (STR ''1.0'', MPointer STR ''1.0''),
       (STR ''1.0.0'', MValue STR ''False''),
       (STR ''0.0.0'', MValue STR ''True''),
       (STR ''0.0'', MPointer STR ''0.0'')],
     Toploc = 1,
     Typed_Mapping = fmap_of_list
      [(STR ''1.1.0'', MTValue TBool),
       (STR ''0.1.0'', MTValue TBool),
       (STR ''1.0'', MTArray 2 (MTValue TBool)),
       (STR ''1.0.0'', MTValue TBool),
       (STR ''0.0.0'', MTValue TBool),
       (STR ''0.0'', MTArray 2 (MTValue TBool))]|)"


lemma reversible_CompMemJustType_imps_same:
```

```
  assumes "CompMemJustType t1 t2"
    and "CompMemJustType t2 t1"
  shows "t1 = t2"
  using assms
proof (induction t1 arbitrary: t2)
  case (MTValue tp)
  then show ?case by simp
next
  case (MTArray len arr)
  then obtain len2 arr2 where a10:"t2 = MTArray len2 arr2"
    using CompMemJustType.simps
    by (metis mtypes.exhaust)

  then show ?case
  proof(cases "len2 = len ∧ arr2 = arr")
    case True
    then show ?thesis using a10 by auto
  next
    case False
    then have "CompMemJustType arr arr2"
      using MTArray.prems(1) a10
      by (metis CompMemTypes_asc mtypes.exhaust mtypes.inject(1) CompMemJustType.simps(1)
CompMemJustType.simps(2))
    moreover have "CompMemJustType arr2 arr" using a10 MTArray.prems False
      by (metis CompMemJustType.simps(1,2) CompMemTypes_asc less_not_refl mtype_size.cases
mtype_size_strict_decrease_array)
    ultimately have "arr = arr2"
      using MTArray.IH by auto
    then show ?thesis using MTArray ‹CompMemJustType arr2 arr› ‹t2 = MTArray len2 arr2› by force
  qed
qed


lemma CompMemTypeSameLocsSameType:
  assumes "MCon (MTArray len arr) mem p'"
    and "CompMemType mem len arr tp1 p' stl1"
    and "CompMemType mem len arr tp2 p' stl1"
  shows "tp2 = tp1"
  using MCon_CompMemJustType_unique[OF CompMemType_imps_CompMemJustType[OF assms(3)]
      CompMemType_imps_CompMemJustType[OF assms(2)]
      CompTypeRemainsMCon[OF assms(3,1)] CompTypeRemainsMCon[OF assms(2,1)]] .

lemma CompMemType_asc_withSharedTarget:
  assumes "CompMemType mem len x (MTArray x' t') p' stl2"
    and "CompMemType mem len x tp1 p' stl1"
    and "MCon (MTArray len x) mem p'"
    and "TypedMemSubPrefPtrs mem x' t' stl2 stl1"
  shows "CompMemType mem x' t' tp1 stl2 stl1" using assms(1,3,4)
proof(induction t' arbitrary:stl2 x')
  case (MTArray x1 t')
  have mc2:"MCon (MTArray x' (MTArray x1 t')) mem stl2" using CompTypeRemainsMCon[OF
MTArray.prems(1,2)] by simp
  have "(∃i<x'. ∃l. accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer l)
        ∧ (l = stl1 ∨ TypedMemSubPrefPtrs mem x1 t' l stl1))"
    using MTArray TypedMemSubPrefPtrs.simps(2)[of mem x' x1 t' stl2 stl1] by simp
  then obtain i l where idef:"i<x'" and ldef:"accessStore (hash stl2 (ShowL_nat i)) mem = Some
(MPointer l) ∧ (l = stl1 ∨ TypedMemSubPrefPtrs mem x1 t' l stl1)" by blast
  have MConexpand:" ∀i<x'.
             (case accessStore (hash stl2 (ShowL_nat i)) mem of None ⇒ False
              | Some (MValue val) ⇒ (case MTArray x1 t' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                  (MTArray x1 t') mem (hash stl2 (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case MTArray x1 t' of MTArray len' arr' ⇒ MCon (MTArray x1 t')
mem loc2 | MTValue Types ⇒ False)) ∧
             ((∃p. accessStore stl2 mem = Some (MPointer p)) ∨ accessStore stl2 mem = None)"
```

```
    using mc2 MTArray.prems MCon.simps(2)[of x' "(MTArray x1 t')" mem stl2] by simp
  have " (accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l = stl1 ∧ MTArray x1 t' =
tp1 ∨ CompMemType mem x1 t' tp1 l stl1))"
  proof(cases "l = stl1")
    case True
    then show ?thesis
      by (metis CompMemTypeSameLocsSameType MTArray.prems(1) assms(2) assms(3) compMemTypes_trns idef
ldef CompMemType.simps(2))
  next
    case False
    then have "TypedMemSubPrefPtrs mem x1 t' l stl1" using ldef by simp
    then show ?thesis
      by (meson CompTypeRemainsMCon MTArray.IH MTArray.prems(1) assms(2) assms(3) compMemTypes_trns
idef ldef CompMemType.simps(2))
  qed
  then show ?case using CompMemType.simps(2)[of mem x' x1 t' tp1 stl2 stl1] idef ldef by blast
next
  case (MTValue x)
  then show ?case
    by (metis CompMemTypeSameLocsSameType TypedMemSubPrefPtrs.simps(1) assms(2) compMemTypes_trns
CompMemType.simps(1))
qed

lemma TypedMemSubPrefOneWay:
  assumes "TypedMemSubPrefPtrs mem x1 t1 stl1 stl2"
    and "CompMemJustType (MTArray x1 t1) (MTArray x2 t2)"
    and "MCon (MTArray x1 t1) mem stl1"
    and "MCon (MTArray x2 t2) mem stl2"
    and "CompMemType mem x1 t1 (MTArray x2 t2) stl1 stl2"
  shows "¬TypedMemSubPrefPtrs mem x2 t2 stl2 stl1"
  using assms
proof(induction t2 arbitrary:x2 stl2)
  case (MTArray x2' t2')
  then show ?case
  proof(induction t1 arbitrary: x1 stl1)
    case (MTArray x1' t1')
    show ?case
    proof
      assume *:"TypedMemSubPrefPtrs mem x2 (MTArray x2' t2') stl2 stl1"
      then have a10:"(∃i<x2. ∃l. accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l =
stl1 ∨ TypedMemSubPrefPtrs mem x2' t2' l stl1))"
        using TypedMemSubPrefPtrs.simps(2)[of mem x2 x2' t2' stl2 stl1] by simp
      have a20:"(∃i<x1. ∃l. accessStore (hash stl1 (ShowL_nat i)) mem = Some (MPointer l) ∧ (l = stl2 ∨
TypedMemSubPrefPtrs mem x1' t1' l stl2))"
        using MTArray(3) TypedMemSubPrefPtrs.simps(2)[of mem x1 x1' t1' stl1 stl2] by simp
      have a30:"∀i<x1.
            (case accessStore (hash stl1 (ShowL_nat i)) mem of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x1' t1' of MTArray l a ⇒ False | MTValue typ ⇒
MCon
                 (MTArray x1' t1') mem (hash stl1 (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x1' t1' of MTArray len' arr' ⇒ MCon (MTArray x1'
t1') mem loc2 | MTValue Types ⇒ False))
          ∧ ((∃p. accessStore stl1 mem = Some (MPointer p)) ∨ accessStore stl1 mem = None)"
        using MTArray MCon.simps(2)[of x1 "MTArray x1' t1'" mem stl1] by simp
      have a40:"∀i<x2.
            (case accessStore (hash stl2 (ShowL_nat i)) mem of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x2' t2' of MTArray l a ⇒ False | MTValue typ ⇒
MCon
                 (MTArray x2' t2') mem (hash stl2 (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x2' t2' of MTArray len' arr' ⇒ MCon (MTArray x2'
t2') mem loc2 | MTValue Types ⇒ False)) ∧
            ((∃p. accessStore stl2 mem = Some (MPointer p)) ∨ accessStore stl2 mem = None)"
        using MTArray MCon.simps(2)[of x2 "MTArray x2' t2'" mem stl2] by simp
      obtain i1 l1 where i1def:"i1<x1 ∧ accessStore (hash stl1 (ShowL_nat i1)) mem = Some (MPointer l1)
```

```
∧ (l1 = stl2 ∨ TypedMemSubPrefPtrs mem x1' t1' l1 stl2)"
        using a20 by blast
      obtain i2 l2 where i2def:"i2<x2 ∧ accessStore (hash stl2 (ShowL_nat i2)) mem = Some (MPointer l2)
∧ (l2 = stl1 ∨ TypedMemSubPrefPtrs mem x2' t2' l2 stl1)"
        using a10 by blast
      show False
      proof(cases "l1 = stl2")
        case t1:True
        then show ?thesis
        proof(cases "l2 = stl1")
          case t2:True
          then have "MCon (MTArray x1' t1') mem stl2" using t1 i2def i1def a30 by fastforce
          have b1:"MCon (MTArray x2' t2') mem stl1" using t1 t2 i2def i1def a40 by fastforce
          have b2:"MCon (MTArray x1 (MTArray x1' t1')) mem stl1" using MTArray by simp


          show ?thesis using b1 MTArray.prems(3,4,5)
            by (metis MConSubTypes <MCon (MTArray x1' t1') mem stl2> CompMemJustType.simps(2))
        next
          case False
          then have "TypedMemSubPrefPtrs mem x2' t2' l2 stl1" using i2def by simp
          moreover have "TypedMemSubPrefPtrs mem x1 (MTArray x1' t1') stl1 l2" using MTArray.prems(2)
i2def
            using MTArray.prems(4,5,6) TypedMemSubPrefPtrs.simps(2) TypedMemSubPrefPtrs_trns by blast
          moreover have "CompMemJustType (MTArray x1 (MTArray x1' t1')) (MTArray x2' t2')" using
MTArray
            by (metis CompMemTypes_asc CompMemJustType.simps(2))
          moreover have "MCon (MTArray x1 (MTArray x1' t1')) mem stl1" using MTArray.prems by auto
          moreover have "MCon (MTArray x2' t2') mem l2" using i2def a40 by fastforce
          moreover have "CompMemType mem x1 (MTArray x1' t1') (MTArray x2' t2') stl1 l2"
            by (meson MTArray.prems(6) compMemTypes_trns i2def CompMemType.simps(2))
          ultimately show ?thesis using MTArray(2)[of l2 x2'] by blast
        qed
      next
        case False
        then show ?thesis
        proof(cases "l2 = stl1")
          case t2:True
          have "TypedMemSubPrefPtrs mem x1' t1' l1 stl2" using False
            using i1def by auto
          then show ?thesis using t2
            by (metis CompMemType.simps(2) MConPtrsMustBeSubLocs_CompMemType MCon_imps_sub_Mcon
MTArray.prems(4,5,6) compMemTypes_trns i2def)
        next
          case f2:False
          then have "TypedMemSubPrefPtrs mem x1' t1' l1 stl2" using i1def False by simp
          moreover have "TypedMemSubPrefPtrs mem x2' t2' l2 stl1" using i2def f2 by simp
          moreover have "CompMemJustType (MTArray x1' t1') (MTArray x2 (MTArray x2' t2'))" using
MTArray
            by (metis CompMemType_imps_CompMemJustType CompMemJustType.simps(2) CompMemType.simps(2))
          moreover have "MCon (MTArray x1' t1') mem l1"
            using a30 i1def by fastforce
          moreover have "CompMemType mem x1' t1' (MTArray x2 (MTArray x2' t2')) l1 stl2"
            by (meson CompMemType_asc_withSharedTarget CompTypeRemainsMCon MTArray.prems(4)
MTArray.prems(6) calculation(1) i1def CompMemType.simps(2))
          ultimately have "¬ TypedMemSubPrefPtrs mem x2 (MTArray x2' t2') stl2 l1"
            using MTArray(1)[of x1' l1] * False
            using MConSubTypes MTArray.prems(4) CompMemJustType.simps(2) MTArray(6)
            by (metis BothMConImpsNotCompMemType CompMemType_asc_withSharedTarget compMemTypes_trns
i1def CompMemType.simps(2))
          then show ?thesis using * False f2
            using MConSubTypes MTArray.prems(4) CompMemJustType.simps(2)
            by (meson CompMemType_imps_CompMemJustType CompMemType_imps_TypedMemSubPrefPtrs
CompTypeRemainsMCon MTArray.prems(1) MTArray.prems(6) compMemTypes_trns i2def CompMemType.simps(2))
```

```
      qed
    qed
  qed

 next
  case (MTValue x)
  then show ?case by fastforce
 qed
next
 case (MTValue x)
 then show ?case
 proof(induction t1 arbitrary:x1 stl1)
  case (MTArray x1 t1)
  then show ?case
   by fastforce
 next
  case (MTValue x)
  then show ?case by fastforce
 qed

qed

lemma TypedMemSubPrefPtrs_imps_notsame:
 assumes "TypedMemSubPrefPtrs m len'' arr'' stl1 prnt"
  and "MCon (MTArray len'' arr'') m stl1"
 shows "stl1 ≠ prnt" using assms
proof(induction arr'' arbitrary:len'' stl1)
 case (MTArray x1 arr'')
 obtain i l where iDef:"i<len'' ∧ accessStore (hash stl1 (ShowL_nat i)) m = Some (MPointer l) ∧ (l =
prnt ∨ TypedMemSubPrefPtrs m x1 arr'' l prnt)"
  using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
 then have mc:"MCon (MTArray x1 arr'') m l" using MTArray.prems(2)
  using MCon_imps_sub_Mcon by blast
 then show ?case
 proof(cases "l= prnt")
  case True
  then show ?thesis using iDef mc
   using MConPtrsMustBeSubLocs MTArray.prems(2) by blast
 next
  case False
  then show ?thesis using MTArray.IH iDef mc
   by (metis CompMemType_imps_CompMemJustType CompMemType_imps_TypedMemSubPrefPtrs MTArray.prems(2)
TypedMemSubPrefOneWay CompMemType.simps(2))
 qed
next
 case (MTValue x)
 then show ?case by force
qed

lemma existingLocation_imps_allLocs:
 assumes "CompMemType m len (MTArray len' arr') subT prnt stl1"
  and "MCon (MTArray len (MTArray len' arr')) m prnt"
 shows "∀ i<len. ∀ l. accessStore (hash prnt (ShowL_nat i)) m = Some (MPointer l) ∧ l≠stl1 ∧
TypedMemSubPrefPtrs m len' arr' l stl1
⟶ CompMemType m len' arr' subT l stl1"
proof intros
 fix i l
 assume asm1:"i < len"
  and asm2:"accessStore (hash prnt (ShowL_nat i)) m = Some (MPointer l) ∧ l ≠ stl1 ∧
TypedMemSubPrefPtrs m len' arr' l stl1"
 show "CompMemType m len' arr' subT l stl1" using assms asm2 asm1
 proof(induction arr' arbitrary:len' l)
  case (MTArray x1 arr')
  then show ?case
```

```
      by (meson CompMemType_asc_withSharedTarget CompTypeRemainsMCon CompMemType.simps(2))
  next
    case (MTValue x)
    then show ?case
      by (meson CompMemType_asc_withSharedTarget CompTypeRemainsMCon CompMemType.simps(2))
  qed
qed


lemma existingLocation_imps_allLocs_same:
  assumes "CompMemType m len t subT prnt stl1"
    and "MCon (MTArray len t) m prnt"
  shows "∀i<len. ∀l. accessStore (hash prnt (ShowL_nat i)) m = Some (MPointer l) ∧ l =stl1
⟶ subT = t"
proof intros
  fix i l
  assume asm1:"i < len"
    and asm2:"accessStore (hash prnt (ShowL_nat i)) m = Some (MPointer l) ∧ l = stl1 "
  show "subT = t" using assms asm2 asm1
  proof(induction t arbitrary:len l)
    case (MTArray x1 arr')

    obtain i'' ptr where ptrDef:"(i''<len ∧ accessStore (hash prnt (ShowL_nat i'')) m = Some (MPointer
ptr)
                                  ∧ (ptr = stl1 ∧ MTArray x1 arr' = subT
                                      ∨ CompMemType m x1 arr' subT ptr stl1))"
      using MTArray.prems(1) unfolding CompMemType.simps(2) by blast
    then show ?case
    proof(cases "ptr = l")
      case True
      then show ?thesis
        using BothMConImpsNotCompMemType CompTypeRemainsMCon MCon_imps_sub_Mcon MTArray.prems(2,3)
ptrDef by blast
    next
      case False
      then have "CompMemType m x1 arr' subT ptr stl1" using ptrDef
        by (simp add: MTArray.prems(3))
      then show ?thesis using ptrDef False
        using CompMemTypeSameLocsSameType MTArray.prems(2,3,4) CompMemType.simps(2) by blast
    qed

  next
    case (MTValue x)
    then show ?case by simp
  qed
qed

lemma CompMemType_imps_in_GetAllMemoryLocations_ptr:
  assumes "MCon (MTArray len2 arr2) mem stl2"
    and "CompMemType mem len2 arr2 (MTArray len3 arr3) stl2 locs"
  shows "∀i<len3. hash locs (ShowL_nat i) ∈ GetAllMemoryLocations (MTArray len2 arr2) stl2 mem" using
assms
proof (induction arr2 arbitrary: len2 stl2)
  case (MTArray x11 x12)
  have "(∃i<len2.
        ∃l. accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer l)
        ∧ (l = locs ∧ MTArray x11 x12 = MTArray len3 arr3 ∨ CompMemType mem x11 x12 (MTArray len3
arr3) l locs))"
    using MTArray.prems(2) CompMemType.simps(2)[of mem len2 x11 x12 "(MTArray len3 arr3)" stl2 locs] by
blast
  then obtain i l where iDef:"i<len2 ∧ accessStore (hash stl2 (ShowL_nat i)) mem = Some (MPointer l)
            ∧ (l = locs ∧ MTArray x11 x12 = MTArray len3 arr3 ∨ CompMemType mem x11 x12 (MTArray
len3 arr3) l locs)" by blast
  then have mc:"MCon (MTArray x11 x12) mem l" using MTArray.prems(1)
```

```
    using MCon_imps_sub_Mcon by blast
  then show ?case
  proof(cases "l = locs")
    case True
    then have st:"MTArray x11 x12 = MTArray len3 arr3" using iDef
      using MTArray.prems(1,2) existingLocation_imps_allLocs_same by blast
    then have "∀i''<len3. hash l (ShowL_nat i'') ∈ {hash st12 (ShowL_nat i)} ∪ GetAllMemoryLocations
(MTArray len3 arr3) l mem"
      using GetAllMemoryLocations_Arry_Contains_Indexs[OF mc] by blast
    then show ?thesis using GetAllMemoryLocations.simps(2)[of len2 "MTArray len3 arr3" st12 mem] st
True iDef by force
  next
    case False
    then have "CompMemType mem x11 x12 (MTArray len3 arr3) l locs" using iDef by simp
    then have "∀i<len3. hash locs (ShowL_nat i) ∈ GetAllMemoryLocations (MTArray x11 x12) l mem"
      using MTArray.IH[OF mc] by blast
    then show ?thesis using GetAllMemoryLocations.simps(2)[of len2 "MTArray x11 x12" st12 mem] iDef
by force
  qed
next
  case (MTValue x2)
  then show ?case using MTValue by auto
qed


lemma typedPrefix_imp_SubPref:
  shows "TypedMemSubPref child parent t ⟶ LSubPrefL2 child parent"
proof(induction t arbitrary:parent)
  case (MTArray x1 t)
  show ?case
  proof intros
    assume *:"TypedMemSubPref child parent (MTArray x1 t)"
    then have **:"(∃i<x1. TypedMemSubPref child (hash parent (ShowL_nat i)) t ∨ child = (hash parent
(ShowL_nat i)))"
      using TypedMemSubPref.simps(2)[of child parent x1 t] by auto
    then show "LSubPrefL2 child parent"
    proof(cases "∃i<x1. child = (hash parent (ShowL_nat i))")
      case True
      then show ?thesis unfolding LSubPrefL2_def by auto
    next
      case False
      then obtain i where "i < x1 ∧ TypedMemSubPref child (hash parent (ShowL_nat i)) t" using ** by
auto
      then have "LSubPrefL2 child (hash parent (ShowL_nat i))" using MTArray by auto

      then show "LSubPrefL2 child parent" unfolding LSubPrefL2_def
        using LSubPrefL2_def Not_Sub_More_Specific by blast
    qed

  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    assume "TypedMemSubPref child parent (MTValue x)"
    then show "LSubPrefL2 child parent" unfolding LSubPrefL2_def by auto
  qed
qed
```

Locations inside element `i < x` of an array are never below index `x`.

```
lemma TypedMemSubPref_not_below_index:
  assumes "TypedMemSubPref l p (MTArray x t)"
    and "LSubPrefL2 l' l"
  shows "¬ LSubPrefL2 l' (hash p (ShowL_nat x))"
```

```
proof
  assume below: "LSubPrefL2 l' (hash p (ShowL_nat x))"
  obtain i where i: "i < x" "TypedMemSubPref l (hash p (ShowL_nat i)) t ∨ l = hash p (ShowL_nat i)"
    using assms(1) by auto
  then have "LSubPrefL2 l (hash p (ShowL_nat i))"
    using typedPrefix_imp_SubPref LSubPrefL2_refl by blast
  then have "LSubPrefL2 l' (hash p (ShowL_nat i))"
    using assms(2) unfolding LSubPrefL2_def by (metis hash_suffixes_associative)
  then have "i = x" using below LSubPrefL2_index_unique by blast
  then show False using i(1) by simp
qed

corollary TypedMemSubPref_index_disjoint:
  "TypedMemSubPref l p (MTArray x t) ⟹ ¬ LSubPrefL2 l (hash p (ShowL_nat x))"
  using TypedMemSubPref_not_below_index LSubPrefL2_refl by blast

corollary TypedMemSubPref_index_disjoint_typed:
  "TypedMemSubPref l p (MTArray x t) ⟹ ¬ TypedMemSubPref l (hash p (ShowL_nat x)) t'"
  using TypedMemSubPref_index_disjoint typedPrefix_imp_SubPref by blast

corollary TypedMemSubPref_hash_not_below_index:
  "TypedMemSubPref l p (MTArray x t) ⟹ ¬ LSubPrefL2 (hash l k) (hash p (ShowL_nat x))"
  using TypedMemSubPref_not_below_index LSubPrefL2_hash by blast

corollary TypedMemSubPref_hash_index_disjoint_typed:
  "TypedMemSubPref l p (MTArray x t) ⟹ ¬ TypedMemSubPref (hash l k) (hash p (ShowL_nat x)) t'"
  using TypedMemSubPref_not_below_index LSubPrefL2_hash typedPrefix_imp_SubPref by blast


lemma CompMemTypeSubTypes_neg:
  shows "¬CompMemType v' x1 t (MTArray x1 t) l destl'"
proof
  assume "CompMemType v' x1 t (MTArray x1 t) l destl'"
  then have a1:"CompMemType v' x1 t (MTArray x1 t) l destl'" by simp
  show False using a1
  proof(induction t arbitrary:x1 l)
    case (MTArray x11 x12)
    then show ?thesis unfolding CompMemType.simps
      by (metis CompMemJustType.simps(2) CompMemType_imps_CompMemJustType MTArray.prems
mtypes.inject(1)
          reversible_CompMemJustType_imps_same)
  next
    case (MTValue x2)
    then show ?thesis by simp
  qed
qed


lemma CompMemType_imps_Mid:
  assumes "CompMemType mem len arr (MTArray x11 x12) pParentPtr p"
  shows "arr = MTArray x11 x12 ∧ (∃i<len. accessStore (hash pParentPtr (ShowL_nat i)) mem = Some
(MPointer p))
       ∨ (∃midP subL subA i.
            CompMemType mem len arr (MTArray subL subA) pParentPtr midP
            ∧ accessStore (hash midP (ShowL_nat i)) mem = Some (MPointer p)
            ∧ i < subL
            ∧ subA = MTArray x11 x12)"
  using assms
proof(induction arr arbitrary:len pParentPtr)
  case (MTArray x1 arr)
  obtain ii ll where iiDef:"ii<len ∧ accessStore (hash pParentPtr (ShowL_nat ii)) mem = Some (MPointer
ll) ∧
(ll = p ∧ MTArray x1 arr = MTArray x11 x12 ∨ CompMemType mem x1 arr (MTArray x11 x12) ll p)"
    using MTArray.prems unfolding CompMemType.simps by blast
```

```
  then show ?case
  proof(cases "MTArray x1 arr = MTArray x11 x12")
    case True
    then have "ll = p" using iiDef
      using CompMemTypeSubTypes_neg by blast
    then show ?thesis using iiDef True by blast
  next
    case False
    then have comp: "CompMemType mem x1 arr (MTArray x11 x12) ll p" using iiDef by blast
    then show ?thesis using MTArray.IH[of x1 ll] iiDef by fastforce
  qed
next
  case (MTValue x)
  then show ?case unfolding CompMemType.simps by simp
qed
```

5.5 Locations above the top location

```
lemma AllPtrsNotTop_MCon:
  assumes "lessThanTopLocs mem"
  shows "∀tl. tl ≥ Toploc mem ∧ MCon t mem ptr⟶
        (¬LSubPrefL2 ptr (ShowL_nat tl))∧
        (¬LSubPrefL2 (ShowL_nat tl) ptr)"
proof intros
  fix tl
  assume *:" Toploc mem ≤ tl ∧ MCon t mem ptr"
  then have ptrMCon:"MCon t mem ptr" by force
  have SomeA:"∃x i. accessStore ptr mem = Some x ∨ accessStore (hash ptr (ShowL_nat i)) mem = Some x"
    using MCon_imps_Some[OF ptrMCon] by blast
  have toplocs:"lessThanTopLocs mem" using assms(1) by blast
  then have noneA:"accessStore (ShowL_nat (Toploc mem)) mem = None
          ∧ (∀locs. accessStore (hash (ShowL_nat (Toploc mem)) locs) mem = None)"
    unfolding lessThanTopLocs_def
    using LSubPrefL2_def by blast
  show "¬ LSubPrefL2 ptr (ShowL_nat tl)"
    using LSubPrefL2_def SomeA hash_suffixes_associative noneA *
    by (metis lessThanTopLocs_def option.discI toplocs)
  show "¬ LSubPrefL2 (ShowL_nat tl) ptr" using LSubPrefL2_def MemLSubPrefL2_specific_imps_general *
    using MemLSubPrefTransitive <¬ LSubPrefL2 ptr (ShowL_nat tl)> by blast
qed


lemma lessThanTop_topChange_MCon:
  assumes "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mo)) ∨ locs = (ShowL_nat
      (Toploc mo)) ⟶ accessStore locs mo = accessStore locs m"
    and "MCon tp mo x2"
    and "¬ LSubPrefL2 (ShowL_nat (Toploc mo)) x2 ∧ ¬ LSubPrefL2 x2 (ShowL_nat (Toploc mo))"
    and "lessThanTopLocs mo"
  shows " MCon tp (m) x2" using assms(2,3)
proof(induction tp arbitrary:x2)
  case (MTArray x1 tp)
  then have old:"0 < x1 ∧ (∀i<x1.
    case accessStore (hash x2 (ShowL_nat i)) mo of None ⇒ False | Some (MValue val) ⇒
(case tp of MTArray l a ⇒ False | MTValue typ ⇒ MCon tp mo (hash x2 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case tp of MTArray len' arr' ⇒ MCon tp mo loc2 | MTValue val ⇒ False)) ∧
(∃p. accessStore x2 mo = Some (MPointer p) ∨ accessStore x2 mo = None)"
    unfolding MCon.simps by simp
  then have "(∃p. accessStore x2 m = Some (MPointer p) ∨ accessStore x2 m = None)"
    using assms(1) MTArray.prems(2) by auto
  moreover have "(∀i<x1.
    case accessStore (hash x2 (ShowL_nat i)) m of None ⇒ False | Some (MValue val) ⇒
(case tp of MTArray l a ⇒ False | MTValue typ ⇒ MCon tp m (hash x2 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case tp of MTArray len' arr' ⇒ MCon tp m loc2 | MTValue val ⇒ False))"
  proof intros
```

```
    fix i
    assume in1:"i<x1"
    then have sameAcc:"accessStore (hash x2 (ShowL_nat i)) mo = accessStore (hash x2 (ShowL_nat i)) m"
      using MTArray.prems(2) assms(1)
      using Mutual_NonSub_SpecificNonSub by presburger
    show " case accessStore (hash x2 (ShowL_nat i)) m of None ⇒ False | Some (MValue val) ⇒
(case tp of MTArray l a ⇒ False | MTValue typ ⇒ MCon tp m (hash x2 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case tp of MTArray len' arr' ⇒ MCon tp m loc2 | MTValue val ⇒ False)"
    proof(cases "accessStore (hash x2 (ShowL_nat i)) mo")
      case None
      then show ?thesis using sameAcc old in1 by auto
    next
      case (Some a)
      then show ?thesis
      proof(cases a)
        case (MValue x1')
        then have "MCon tp mo (hash x2 (ShowL_nat i))"
          using sameAcc old in1 Some
          using MCon_imps_sub_Mcon MTArray.prems(1) by blast
        moreover have "∃v. tp = MTValue v" using calculation Some MValue by (cases tp, simp+)
        ultimately show ?thesis using sameAcc Some MValue
          by (metis MTArray.IH MTArray.prems(2) mtypes.simps(6) MemLSubPrefL2_specific_imps_general
memoryvalue.simps(5) Option.option.simps(5) Not_Sub_More_Specific)
      next
        case (MPointer x2')
        then have mcOld:"MCon tp mo x2'" using old Some in1 by (cases tp, auto)

        moreover have "¬ LSubPrefL2 (ShowL_nat (Toploc mo)) x2' ∧ ¬ LSubPrefL2 x2' (ShowL_nat (Toploc
mo))"
          using AllPtrsNotTop_MCon calculation assms(4) by (meson le_refl)
        ultimately have "MCon tp m x2'" using MTArray.IH[of x2'] by simp
        moreover have "∃l a. tp = MTArray l a" using Some MPointer old
          by (metis MTArray.prems(1) memoryvalue.distinct(1) in1 mcon_accessStore option.inject)
        ultimately show ?thesis using in1 sameAcc Some MPointer
          by (metis mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5))
      qed
    qed
  qed
  moreover have "0 < x1" using old by simp
  ultimately show ?case unfolding MCon.simps by simp
next
  case (MTValue x)
  then show ?case using assms(1) by auto
qed


lemma MCon_memory_transfer:
  assumes "MCon t mem_old ptr_loc"
    and "¬ LSubPrefL2 ptr_loc toploc_new ∧ ¬ LSubPrefL2 toploc_new ptr_loc"
    and "∀len arr loc. t = MTArray len arr ∧ TypedMemSubPrefPtrs mem_old len arr ptr_loc loc
         ⟶ ¬ LSubPrefL2 loc toploc_new ∧ ¬ LSubPrefL2 toploc_new loc"
    and "∀loc. ¬ LSubPrefL2 loc toploc_new ∨ loc = toploc_new
         ⟶ accessStore loc mem_new = accessStore loc mem_old"
  shows "MCon t mem_new ptr_loc"
  using assms(1,2,3)
proof(induction t arbitrary:ptr_loc)
  case (MTArray x1 struct)
  have "x1 > 0" using MTArray.prems(1) unfolding MCon.simps
    using bot_nat_0.not_eq_extremum by fastforce
  moreover have "(∃p. accessStore ptr_loc mem_new = Some (MPointer p)) ∨ accessStore ptr_loc mem_new
= None"
    using MTArray.prems(1,2) unfolding MCon.simps using assms(3,4) calculation(1) by simp
  moreover have "∀i<x1.
    (case accessStore (hash ptr_loc (ShowL_nat i)) mem_new of None ⇒ False
```

```
    | Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct mem_new
(hash ptr_loc (ShowL_nat i)))
    | Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct mem_new loc2 | MTValue
val ⇒ False))"
  proof intros
    fix i
    assume *:"i<x1"
    then have notSub:"¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) toploc_new ∨ (hash ptr_loc (ShowL_nat
i)) = toploc_new"
      using MTArray.prems(2) Mutual_NonSub_SpecificNonSub by auto
    show "(case accessStore (hash ptr_loc (ShowL_nat i)) mem_new of None ⇒ False
    | Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct mem_new
(hash ptr_loc (ShowL_nat i)))
    | Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct mem_new loc2 | MTValue
val ⇒ False))"
    proof(cases struct)
      case (MTArray x11' x12')
      obtain loc2 where loc2Def:"accessStore (hash ptr_loc (ShowL_nat i)) mem_old = Some (MPointer
loc2)
                        ∧ MCon struct mem_old loc2"
        using MTArray.prems(1) *
        by (metis MConArrayPointers MCon_imps_sub_Mcon MTArray calculation(1))
      then have o1:"∀loc. TypedMemSubPrefPtrs mem_old x1 struct ptr_loc loc ⟶
                ¬ LSubPrefL2 loc toploc_new ∧ ¬ LSubPrefL2 toploc_new loc"
        using MTArray.prems by simp
      have "∀len arr loc. struct = MTArray len arr ∧ TypedMemSubPrefPtrs mem_old len arr loc2 loc ⟶
          ¬ LSubPrefL2 loc toploc_new ∧ ¬ LSubPrefL2 toploc_new loc"
        using o1 loc2Def * by fastforce
      moreover have "TypedMemSubPrefPtrs mem_old x1 struct ptr_loc loc2"
        using o1 loc2Def *
        using MTArray TypedMemSubPrefPtrs.simps(2) by blast
      moreover have "¬ LSubPrefL2 loc2 toploc_new ∧ ¬ LSubPrefL2 toploc_new loc2"
        using o1 calculation assms by simp
      ultimately have "MCon struct mem_new loc2" using MTArray.IH[of loc2] loc2Def by simp
      moreover have "accessStore (hash ptr_loc (ShowL_nat i)) mem_new = Some (MPointer loc2) "
        using loc2Def * notSub assms(4) by simp
      ultimately show ?thesis
        using MTArray by force
    next
      case (MTValue x2)
      then obtain v where vDef:"accessStore (hash ptr_loc (ShowL_nat i)) mem_old = Some (MValue v)
                          ∧ MCon (MTValue x2) mem_old (hash ptr_loc (ShowL_nat i))"
        using * MTArray.prems(1)
        by (meson MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val)
      then have "accessStore (hash ptr_loc (ShowL_nat i)) mem_new = Some (MValue v)"
        using notSub assms(4) by simp
      then show ?thesis using vDef MTValue unfolding MCon.simps
        using notSub assms(4) by fastforce
    qed
  qed
  ultimately show ?case unfolding MCon.simps by simp
next
  case (MTValue x)
  then show ?case using assms(4) unfolding MCon.simps by simp
qed
```

5.6 Preservation of type relations by copying

```
lemma cpm2mCompMemTypeOld_imps_CompMemType:
  assumes " CompMemType mem' len x tp2 stl1 stl2"
    and "∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs m'"
    and "¬ LSubPrefL2 stl1 l"
    and "∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None"
    and "MCon (MTArray len x) mem' stl1"
```

```
  shows " CompMemType m' len x tp2 stl1 stl2" using assms
proof(induction x arbitrary:stl1 len)
  case (MTArray x1 tp1)
  have c10:"(∃ i<len. ∃ l. accessStore (hash stl1 (ShowL_nat i)) mem' = Some (MPointer l)
           ∧ ((l = stl2) ∧ MTArray x1 tp1 = tp2 ∨ CompMemType mem' x1 tp1 tp2 l stl2))"
    using MTArray(2) CompMemType.simps(2)[of mem' len x1 tp1 tp2 stl1 stl2] using hash_inequality by
auto
  then obtain i l' where idef:"i<len ∧ accessStore (hash stl1 (ShowL_nat i)) mem' = Some (MPointer l')
∧ ((l'
      = stl2 ) ∧ MTArray x1 tp1 = tp2 ∨ CompMemType mem' x1 tp1 tp2 l' stl2)"
    using c10 by auto
  then have c20:"accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l')" using MTArray(3, 4)
LSubPrefL2_def
    by (metis MTArray(3,5) not_None_eq)
  have "∀ i<len.
          (case accessStore (hash stl1 (ShowL_nat i)) mem' of None ⇒ False
           | Some (MValue val) ⇒ (case MTArray x1 tp1 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
               (MTArray x1 tp1) mem' (hash stl1 (ShowL_nat i)))
           | Some (MPointer loc2) ⇒ (case MTArray x1 tp1 of MTArray len' arr' ⇒ MCon (MTArray x1 tp1)
mem' loc2 | MTValue Types ⇒ False)) ∧
          ((∃ p. accessStore stl1 mem' = Some (MPointer p)) ∨ accessStore stl1 mem' = None)"
    using MTArray(6) MCon.simps(2)[of len "(MTArray x1 tp1)" mem' stl1] by auto
  then have mcond:"MCon (MTArray x1 tp1) mem' l'" using MTArray(6) using idef c10
    by (metis (no_types, lifting) mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5))
  have "¬ LSubPrefL2 l' l"
    by (metis MCon_imps_Some LSubPrefL2_def assms(4) Not_Sub_More_Specific mcond not_None_eq)
  then have "CompMemType mem' x1 tp1 tp2 l' stl2 ⟶ CompMemType m' x1 tp1 tp2 l' stl2"
    using MTArray(1)[of x1 l'] MTArray(3,5) mcond by auto

  moreover have "((l' = stl2) ∧ MTArray x1 tp1 = tp2 ∨ CompMemType mem' x1 tp1 tp2 l' stl2)" using
c10 idef by simp
  ultimately have "(l' = stl2) ∧ MTArray x1 tp1 = tp2 ∨ CompMemType m' x1 tp1 tp2 l' stl2" using
MTArray(1) MTArray(3,5) mcond by auto

  then show ?case using idef c10 c20
    using MTArray.prems(1) by auto
next
  case (MTValue x)
  then show ?case using TypedMemSubPrefPtrs.simps(1) by auto
qed

lemma inv_cpm2mTPrefOld_imps_TPref:
  assumes "¬TypedMemSubPrefPtrs mem' len x stl1 stl2"
    and "∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs m'"
    and "¬ LSubPrefL2 stl1 l"
    and "∀ loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None"
    and "MCon (MTArray len x) mem' stl1"
  shows "¬TypedMemSubPrefPtrs m' len x stl1 stl2" using assms
proof(induction x arbitrary:stl1 len)
  case (MTArray x1 tp1)
  show ?case
  proof
    assume *:"TypedMemSubPrefPtrs m' len (MTArray x1 tp1) stl1 stl2"
    then have a10:"(∃ i<len. ∃ l. accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l) ∧ (l =
stl2 ∨ TypedMemSubPrefPtrs m' x1 tp1 l stl2))"
      using TypedMemSubPrefPtrs.simps(2)[of m' len x1 tp1 stl1 stl2] by blast
    have a20:"(∀ i<len. ¬(∃ l. accessStore (hash stl1 (ShowL_nat i)) mem' = Some (MPointer l) ∧
       (l = stl2 ∨ TypedMemSubPrefPtrs mem' x1 tp1 l stl2)))" using MTArray(2) * by auto

    then obtain i l' where idef:"i<len ∧ accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l')
∧ (l' =
       stl2 ∨ TypedMemSubPrefPtrs m' x1 tp1 l' stl2)" using a10 by auto
    then have c20:"accessStore (hash stl1 (ShowL_nat i)) mem' = Some (MPointer l')" using MTArray(3, 4)
LSubPrefL2_def
```

```
      by (metis MConArrayPointers MTArray.prems(5) a10 assms(4) gr_zeroI less_nat_zero_code
option.discI)

    have "∀i<len.
            (case accessStore (hash stl1 (ShowL_nat i)) mem' of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x1 tp1 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                 (MTArray x1 tp1) mem' (hash stl1 (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x1 tp1 of MTArray len' arr' ⇒ MCon (MTArray x1
tp1) mem' loc2 | MTValue Types ⇒ False))"
      using MTArray(6) MCon.simps(2)[of len "MTArray x1 tp1" mem' stl1] by auto
    then have c30:"MCon (MTArray x1 tp1) mem' l'" using MTArray(6) using idef c20
      by (metis (no_types, lifting) mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5))
    moreover have "¬ LSubPrefL2 l' l" using MTArray(4,5) c20 idef c30
      by (metis MCon_imps_Some LSubPrefL2_def Not_Sub_More_Specific not_None_eq)
    ultimately show False using MTArray
      using a20 c20 idef by auto
  qed
next
  case (MTValue x)
  then show ?case by simp
qed


lemma selfPoint_imps_TypedMemSubPref:
  assumes "∀l l'. TypedMemSubPref l ld (MTArray x t) ∧ accessStore l v' = Some (MPointer l') ⟶ l' =
l"
    and "TypedMemSubPrefPtrs v' x t ld x3"
  shows "TypedMemSubPref x3 ld (MTArray x t)" using assms(1,2)
proof(induction t arbitrary:ld x)
  case (MTArray x1 t)
  then have "∃i<x. ∃l. accessStore (hash ld (ShowL_nat i)) v' = Some (MPointer l) ∧ (l = x3 ∨
TypedMemSubPrefPtrs v' x1 t l x3)"
    using TypedMemSubPrefPtrs.simps(2)[of v' x x1 t ld x3] by blast
  then obtain i l where idef:"i<x ∧ accessStore (hash ld (ShowL_nat i)) v' = Some (MPointer l) ∧ (l =
x3 ∨ TypedMemSubPrefPtrs v' x1 t l x3)" by blast
  then have ldef:"l = (hash ld (ShowL_nat i))" using MTArray(2)
    by (metis TypedMemSubPref.simps(2))
  then show ?case
  proof(cases "x3 = l")
    case True
    then show ?thesis using ldef idef by auto
  next
    case False
    have "∀la l'. TypedMemSubPref la l (MTArray x1 t) ∧ accessStore la v' = Some (MPointer l') ⟶ l'
= la" using ldef idef MTArray.prems(1)
      using moreSpecificTypedSubPref by blast
    then have "(TypedMemSubPref x3 l (MTArray x1 t))" using MTArray.IH[of l x1] idef False by blast
    then show ?thesis using TypedMemSubPref.simps(2)[of x3 ld x "MTArray x1 t"] using ldef idef by
blast
  qed
next
  case (MTValue x)
  then show ?case by auto
qed

lemma CompMemType_preservation_induction:
  assumes store_preserved: "∀locs. ¬ LSubPrefL2 locs hash_loc ⟶ accessStore locs v' = accessStore
locs v''"
    and no_prefix_conflict: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l hash_loc"
    and store_consistency: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =
accessStore l v''"
    and type_exists: "CompMemType v' x t (MTArray x11 x12) destl destl' ∧ (∀i<x11. accessTypeStore
(hash destl' (ShowL_nat i)) v' = Some x12)"
    and self_pointers: "∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
```

```
(MPointer l') ⟶ l' = l"
    and type_preserved: "∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore (hash
destl' (ShowL_nat i)) v''"
  shows "CompMemType v'' x t (MTArray x11 x12) destl destl' ∧ (∀i<x11. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some x12)"
  using assms(1,2,3,4,5)
proof(induction t arbitrary: x destl)
  case (MTArray x1 t)
  obtain i l where idef:"i<x ∧ accessStore (hash destl (ShowL_nat i)) v' = Some (MPointer l) ∧
                    (l = destl' ∧ MTArray x1 t = (MTArray x11 x12) ∨ CompMemType v' x1 t (MTArray x11
x12) l destl') ∧
                        (∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
    using MTArray.prems unfolding CompMemType.simps by blast
  then have tps:"TypedMemSubPref (hash destl (ShowL_nat i)) destl (MTArray x (MTArray x1 t))" by auto
  then have self:"l = (hash destl (ShowL_nat i))" using idef MTArray.prems by blast
  then have noSub:"¬ LSubPrefL2 l hash_loc" using idef
    using MTArray.prems(2) tps by presburger
  have sameACC:" accessStore (hash destl (ShowL_nat i)) v'' = accessStore (hash destl (ShowL_nat i)) v'"
using assms(1) MTArray.prems(2) idef by auto
  then show ?case
  proof(cases "MTArray x1 t = (MTArray x11 x12)")
    case True
    then have "¬CompMemType v' x1 t (MTArray x11 x12) l destl'" using CompMemTypeSubTypes_neg by auto
    then have "l = destl'" using idef by simp
    moreover have "accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl (ShowL_nat i))
v''"
      using True tps self noSub MTArray.prems(3) by blast
    moreover have "∀i<x1. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore (hash
destl' (ShowL_nat i)) v''"
      using MTArray.prems(2) assms(6) True by simp
    ultimately show ?thesis using True idef self
      using MTArray.prems(3) tps by auto
  next
    case False
    then have "CompMemType v' x1 t (MTArray x11 x12) l destl'" using idef by auto
    then have cc0:"CompMemType v' x1 t (MTArray x11 x12) l destl' ∧
                (∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
      using idef by simp
    moreover have cc1:"∀la. TypedMemSubPref la l (MTArray x1 t) ⟶ accessStore la v' = accessStore
la v''"
      using MTArray.prems(3) Not_Sub_More_Specific_more_speific self idef tps by blast
    moreover have cc2:"∀la l'. TypedMemSubPref la l (MTArray x1 t) ∧ accessStore la v' = Some
(MPointer l') ⟶ l' = la"
      using MTArray.prems(5) tps self Not_Sub_More_Specific_more_speific idef by blast
    moreover have "∀la. TypedMemSubPref la l (MTArray x1 t) ⟶ ¬ LSubPrefL2 la hash_loc" using self
idef MTArray.prems(2) by auto
    ultimately have cc9:"CompMemType v'' x1 t (MTArray x11 x12) l destl' ∧ (∀i<x11. accessTypeStore
(hash destl' (ShowL_nat i)) v'' = Some x12)"
      using MTArray.IH[OF assms(1) _ cc1 cc0 cc2] noSub self by blast
    then show ?thesis using cc9 idef self sameACC unfolding CompMemType.simps by auto
  qed
next
  case (MTValue x')
  have a0:"CompMemType v' x (MTValue x') (MTArray x11 x12) destl destl'"
    using MTValue by blast
  then show ?case using a0 unfolding CompMemType.simps by simp

qed


lemma CompMemTypeValue_preservation_induction:
  assumes store_preserved: "∀locs. ¬ LSubPrefL2 locs hash_loc ⟶ accessStore locs v' = accessStore
locs v''"
    and no_prefix_conflict: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l hash_loc"
```

```
    and store_consistency: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =
accessStore l v''"
    and type_exists: "accessTypeStore destl' v' = Some (MTValue ttt) ∧ CompMemType v' x t (MTValue ttt)
destl destl'"
    and self_pointers: "∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
    and type_preserved: "accessTypeStore destl' v' = accessTypeStore destl' v''"
  shows "CompMemType v'' x t (MTValue ttt) destl destl' ∧ accessTypeStore destl' v'' = Some (MTValue
ttt)"
  using assms(1,2,3,4,5)
proof(induction t arbitrary: x destl)
  case (MTArray x1 t)
  obtain i l where idef:"accessTypeStore destl' v' = Some (MTValue ttt) ∧ i<x ∧ accessStore (hash
destl (ShowL_nat i)) v' = Some (MPointer l) ∧
                    (l = destl' ∧ MTArray x1 t = (MTValue ttt) ∨ CompMemType v' x1 t (MTValue ttt) l
destl')"
    using MTArray.prems(4) unfolding CompMemType.simps by blast
  then have tps:"TypedMemSubPref (hash destl (ShowL_nat i)) destl (MTArray x (MTArray x1 t))" by auto
  then have self:"l = (hash destl (ShowL_nat i))" using idef MTArray.prems(5) by blast
  then have noSub:"¬ LSubPrefL2 l hash_loc" using idef
    using MTArray.prems(2) tps by presburger
  have sameACC:" accessStore (hash destl (ShowL_nat i)) v'' = accessStore (hash destl (ShowL_nat i)) v'"
using assms(1) MTArray.prems(2) idef by auto

  then have "CompMemType v' x1 t (MTValue ttt) l destl'" using idef by simp
  then have cc0:" accessTypeStore destl' v' = Some (MTValue ttt) ∧ CompMemType v' x1 t (MTValue ttt) l
destl'" using idef by blast
  moreover have cc1:"∀la. TypedMemSubPref la l (MTArray x1 t) ⟶ accessStore la v' = accessStore la
v''"
    using MTArray.prems(3) Not_Sub_More_Specific_more_speific self idef tps by blast
  moreover have cc2:"∀la l'. TypedMemSubPref la l (MTArray x1 t) ∧ accessStore la v' = Some (MPointer
l') ⟶ l' = la"
    using MTArray.prems(5) tps self Not_Sub_More_Specific_more_speific idef by blast
  moreover have "∀la. TypedMemSubPref la l (MTArray x1 t) ⟶ ¬ LSubPrefL2 la hash_loc" using self
idef MTArray.prems(2) by auto
  ultimately have cc9:"CompMemType v'' x1 t (MTValue ttt) l destl' ∧ accessTypeStore destl' v'' = Some
(MTValue ttt)"
    using MTArray.IH[OF assms(1) _ cc1 cc0 cc2] noSub self by blast
  then show ?case using cc9 idef self sameACC unfolding CompMemType.simps by auto
next
  case (MTValue x')
  have a0:"accessTypeStore destl' v' = Some (MTValue ttt) ∧ CompMemType v' x (MTValue x') (MTValue ttt)
destl destl'"
    using MTValue(4) by blast
  then have cc0:"accessTypeStore destl' v'' = Some (MTValue ttt)"
    using assms(6) by metis
  show ?case using a0 cc0 unfolding CompMemType.simps by simp
qed


lemma CompMemType_extend:
  assumes comp_exists: "∃t''. CompMemType v'' x t t'' destl destl' ∧ accessTypeStore destl' v'' = Some
t''"
  shows "∃t''. CompMemType v'' (Suc x) t t'' destl destl' ∧ accessTypeStore destl' v'' = Some t''"
  using assms
proof(cases t)
  case (MTArray x11 x12)
  have cc:"∃t''. ∃i< x.
 ∃l. accessStore (hash destl (ShowL_nat i)) v'' = Some (MPointer l) ∧
     (l = destl' ∧ MTArray x11 x12 = t'' ∨ CompMemType v'' x11 x12 t'' l destl') ∧ accessTypeStore
destl' v'' = Some t''"
    using assms MTArray by auto
  then have "∃t''. CompMemType v'' (Suc x) (MTArray x11 x12) t'' destl destl'"
    unfolding CompMemType.simps
```

```
    using less_Suc_eq by blast
  then show ?thesis using MTArray cc
    using less_Suc_eq by auto
next
  case (MTValue x2)
  then show ?thesis using assms
    using less_Suc_eq by auto
qed

lemma CompMemType_extend_array:
  assumes comp_exists: "CompMemType v'' x t (MTArray x11' x12') destl destl'
                ∧ (∀i<x11'. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12')"
  shows "CompMemType v'' (Suc x) t (MTArray x11' x12') destl destl'
                ∧ (∀i<x11'. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12')"
  using assms
proof(cases t)
  case (MTArray x11 x12)
  then have "CompMemType v'' x (MTArray x11 x12) (MTArray x11' x12') destl destl'
             ∧ (∀i<x11'. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12')" using assms by
simp
  then have cc:"(∃i<x. ∃l. accessStore (hash destl (ShowL_nat i)) v'' = Some (MPointer l) ∧
            (l = destl' ∧ MTArray x11 x12 = MTArray x11' x12' ∨ CompMemType v'' x11 x12 (MTArray x11'
x12') l destl')) ∧
  (∀i<x11'. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12')"
    unfolding CompMemType.simps by simp
  then have "∃t''. CompMemType v'' (Suc x) (MTArray x11 x12) t'' destl destl'"
    unfolding CompMemType.simps
    using less_Suc_eq by blast
  then show ?thesis using MTArray cc
    using less_Suc_eq by auto
next
  case (MTValue x2)
  then show ?thesis using assms
    using less_Suc_eq by auto
qed

end
```

6 Type consistency for storage (TypeSafe_Storage)

```
theory TypeSafe_Storage
  imports TypeSafe_Base_Types
begin
```

Size measure on storage types, used to rule out impossible recursion branches.

```
fun stype_size :: "stypes ⇒ nat" where
  "stype_size (STValue _) = 1"
| "stype_size (STArray _ S) = 1 + stype_size S"
| "stype_size (STMap _ S) = 1 + stype_size S"
```

6.1 Storage type conformity

The SCon function works similarly to the MCon function and searches a given storage starting at location (loc) to ensure all sub locations conform to their given types. There are two main differences to MCon. 1. The support for the STMap type. 2. There is no need to look up pointer locations when indexing through arrays, storage does not support pointers in solidity and so a direct lookup can be performed.

```
primrec SCon :: "stypes ⇒ location ⇒ storageT ⇒bool"
  where
    "SCon (STValue typ) loc fm= ((typeCon typ (accessStorage typ loc fm)))"
  |"SCon (STArray len arr) loc fm = (∀i<len. SCon arr (hash loc (ShowL_nat i)) fm)"
  |"SCon (STMap fromTyp toTyp) loc fm = (∀i::String.literal. (typeCon fromTyp i) ⟶ SCon toTyp (hash
loc i) fm)"
```

```
lemma subSCon:
  assumes "SCon (STArray (Suc x) t') srcl srcMem"
  shows "SCon (STArray (x) t') srcl srcMem" using assms
proof(induction t' arbitrary: x srcl)
  case (STArray x1 t')
  then show ?case by auto
next
  case (STMap x1 t')
  then show ?case by simp
next
  case (STValue x)
  then show ?case by simp
qed


primrec TypedStoSubpref :: "location ⇒ location ⇒ stypes ⇒ bool"
  where
    "TypedStoSubpref child parent (STValue typ) = (child = parent)"
  | "TypedStoSubpref child parent (STArray len arr) = (child = parent ∨
                                                       (∃i<len. TypedStoSubpref child (hash parent
(ShowL_nat i)) arr))"
  | "TypedStoSubpref child parent (STMap fromTyp toTyp) = ((child = parent)
                                ∨ (∃i::String.literal. (typeCon fromTyp i) ∧ (TypedStoSubpref child
(hash parent i) toTyp)))"


lemma TypedStoSubpref_b:
  assumes "TypedStoSubpref l1 l2 t"
  shows "(∃a. hash l2 a = l1) ∨ l1 = l2" using assms
proof(induction t arbitrary:l2)
  case (STArray x1 t)
  then have a1:"(l1 = l2 ∨ (∃i<x1. TypedStoSubpref l1 (hash l2 (ShowL_nat i)) t))" using
TypedStoSubpref.simps(2)[of l1 l2 x1 t] by simp
  then show ?case
  proof(cases "l1 = l2")
    case True
    then show ?thesis by simp
  next
    case False
    then have "∃i<x1. TypedStoSubpref l1 (hash l2 (ShowL_nat i)) t" using a1 by simp
    then show ?thesis by (metis STArray.IH hash_suffixes_associative)
  qed
next
  case (STMap x1 t)
  then have a1:" (l1 = l2 ∨ (∃i. typeCon x1 i ∧ TypedStoSubpref l1 (hash l2 i) t))" using
TypedStoSubpref.simps(3)[of l1 l2 x1 t ] by auto
  then show ?case
  proof(cases "l1 = l2")
    case True
    then show ?thesis by auto
  next
    case False
    then have "(∃i. typeCon x1 i ∧ TypedStoSubpref l1 (hash l2 i) t)" using a1 by auto
    then show ?thesis using STMap by (metis hash_suffixes_associative)
  qed

next
  case (STValue x)
  then show ?case by simp

qed
```

```
lemma TypedStoSubpref_imp_LSubPrefL2:
  shows "TypedStoSubpref child parent t ⟶ LSubPrefL2 child parent"
  unfolding LSubPrefL2_def using TypedStoSubpref_b by blast

lemma neg_LSubPrefL2_imps_neg_TypedStoSubpref:
  assumes "¬ LSubPrefL2 child prnt"
  shows "¬ TypedStoSubpref child prnt t"
  using assms TypedStoSubpref_imp_LSubPrefL2 by blast

lemma stoMoreSpecificTypedSubpref:
  assumes "∀l l'. TypedStoSubpref l destl (STArray x t) ∧ accessStore l v' = Some (MPointer l') ⟶ l'
= l"
  shows "∀i<x. ∀l l'. TypedStoSubpref l (hash destl (ShowL_nat i)) t ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
proof (intro allI impI, elim conjE)
  fix i l l'
  assume i_lt: "i < x"
    and sub: "TypedStoSubpref l (hash destl (ShowL_nat i)) t"
    and acc: "accessStore l v' = Some (MPointer l')"
  have "TypedStoSubpref l destl (STArray x t)"
    using TypedStoSubpref.simps(2)[of l destl x t] sub i_lt by auto
  with assms acc show "l' = l" by auto
qed


lemma TypedStoSubpref_hashes:
  shows "TypedStoSubpref l2' (hash loc kv) t ⟶ l2' ≠ loc" using TypedStoSubpref_b
  by (metis hash_inequality hash_suffixes_associative)

lemma TypedStoSubpref_sameLoc:
  shows "TypedStoSubpref l1 l1 tp"
proof(cases tp)
  case (STArray x11 x12)
  then show ?thesis by simp
next
  case (STMap x21 x22)
  then show ?thesis by simp
next
  case (STValue x3)
  then show ?thesis by simp
qed

lemma typedStoSub_imps_negInv:
  assumes "short ≠ short'"
  shows "TypedStoSubpref short short' t' ⟶ ¬ TypedStoSubpref short' short t"
proof
  assume *:"TypedStoSubpref short short' t'"
  show "¬ TypedStoSubpref short' short t"
  proof
    assume **:"TypedStoSubpref short' short t"
    obtain a where a10:"(hash short' a = short)" using * assms TypedStoSubpref_b by blast
    obtain a' where a20:"(hash short a' = short')" using ** assms TypedStoSubpref_b by blast
    show False using a10 a20
      by (metis "*" TypedStoSubpref_hashes)
  qed
qed

primrec subStoTp :: "stypes ⇒ stypes ⇒ bool"
  where
    "subStoTp child (STValue typ) = (child = (STValue typ))"
  | "subStoTp child (STArray len arr) = (child = (STArray len arr) ∨ (subStoTp child arr))"
  | "subStoTp child (STMap fromTyp toTyp) = ((child = (STMap fromTyp toTyp)) ∨ (subStoTp child toTyp))"
```

```
primrec CompStoType :: "stypes ⇒ stypes ⇒ location ⇒ location ⇒ bool"
  where
    "CompStoType (STValue typ) childtp parentloc childloc= (childtp = (STValue typ) ∧ (parentloc =
childloc))"
  | "CompStoType (STArray len arr) childtp parentloc childloc = (((childtp = STArray len arr) ∧
parentloc = childloc) ∨
                                                                (∃i<len. CompStoType arr childtp (hash
parentloc (ShowL_nat i)) childloc))"
  | "CompStoType (STMap fromTyp toTyp) childtp parentloc childloc = ( (childtp = STMap fromTyp toTyp ∧
parentloc = childloc) ∨ (∃i::String.literal.
                                                                  (typeCon fromTyp i ∧ (CompStoType
toTyp childtp (hash parentloc i) childloc))))"

lemma CompStoType_imps_TypedStoSubpref:
  assumes "CompStoType t tp l'' l"
  shows "TypedStoSubpref l l'' t" using assms
proof(induction t arbitrary:l'')
  case (STArray x1 t)
  then show ?case by auto
next
  case (STMap x1 t)
  then show ?case by auto
next
  case (STValue x)
  then have "l'' = l" unfolding CompStoType.simps by simp
  then show ?case by simp
qed


lemma CompStoType_sameLocNdTyp:
  shows "CompStoType tp1 tp1 l1 l1"
proof(cases tp1)
  case (STArray x11 x12)
  then show ?thesis by simp
next
  case (STMap x21 x22)
  then show ?thesis by simp
next
  case (STValue x3)
  then show ?thesis by simp
qed

lemma CompStoType_imp_subType:
  shows "loc ≠ loc' ∧ CompStoType t1 t2 loc loc' ⟶ (∃t. subStoTp t1 t ∧ t = t1)"
proof(induction t1)
  case (STArray x1 t1)
  then show ?case by simp
next
  case (STMap x1 t1)
  then show ?case by simp
next
  case (STValue x)
  then show ?case by simp
qed


lemma CompStoType_imps_subloc:
  shows "loc ≠ l2' ∧ CompStoType t1 t2 loc l2' ⟶ (∃i. hash loc i = l2')"
proof(induction t1 arbitrary:loc)
  case (STArray x1 t1)
  show ?case
  proof intros
    assume *:"loc ≠ l2' ∧ CompStoType (STArray x1 t1) t2 loc l2'"
    then have a1:"(∃i<x1. CompStoType t1 t2 (hash loc (ShowL_nat i)) l2')"
```

```
      using CompStoType.simps(2)[of x1 t1 t2 loc l2'] by auto
    then show "(∃ i. hash loc i = l2')" by (metis STArray hash_suffixes_associative)
  qed
next
  case (STMap x1 t1)
  show ?case
  proof intros
    assume *:"loc ≠ l2' ∧ CompStoType (STMap x1 t1) t2 loc l2'"
    then have a1:"(∃ i. typeCon x1 i ∧ CompStoType t1 t2 (hash loc i) l2')"
      using CompStoType.simps(3)[of x1 t1 t2 loc l2'] by auto
    then show " (∃ i. hash loc i = l2')" by (metis STMap hash_suffixes_associative)

  qed
next
  case (STValue x)
  then show ?case by auto
qed

lemma CompStoType_sameLoc_sameType:
  assumes "CompStoType t1 t2 l1 l1"
  shows "t1 = t2"
proof(cases t1)
  case (STArray x11 x12)
  then have "CompStoType (STArray x11 x12) t2 l1 l1" using STArray assms by auto
  then have "(t2 = STArray x11 x12 ∧ l1 = l1 ∨ (∃ i<x11. CompStoType x12 t2 (hash l1 (ShowL_nat i))
l1))" using CompStoType.simps(2)[of x11 x12 t2 l1 l1] by simp
  then show ?thesis by (metis CompStoType_imps_subloc STArray hash_inequality
hash_suffixes_associative)
next
  case (STMap x21 x22)
  then show ?thesis using assms by (metis CompStoType_imps_subloc hash_inequality
hash_suffixes_associative CompStoType.simps(3))
next
  case (STValue x3)
  then show ?thesis using assms by simp
qed


lemma comp_stotype_size_le:
  "CompStoType t1 t2 l1 l2 ⟹ stype_size t2 ≤ stype_size t1"
proof (induction t1 arbitrary: t2 l1 l2)
  case (STArray x1 t1)
  then show ?case by fastforce
next
  case (STMap x1 t1)
  then show ?case by fastforce
next
  case (STValue x)
  then show ?case by simp
qed


lemma comp_stotype_same_type_same_loc:
  shows "CompStoType t t l1 l2 ⟹ l1 = l2"
proof (induction t arbitrary: l1 l2)
  case (STArray x1 t)
  then have a1:"STArray x1 t = STArray x1 t ∧ l1 = l2 ∨ (∃ i<x1. CompStoType t (STArray x1 t) (hash l1
(ShowL_nat i)) l2)"
    unfolding CompStoType.simps by blast
  show "l1 = l2"
  proof (cases "l1 = l2")
    case True
    then show ?thesis by blast
  next
```

```
    case False
    then obtain i where idef: "i<x1 ∧ CompStoType t (STArray x1 t) (hash l1 (ShowL_nat i)) l2"
      using a1 by blast
    then have "stype_size (STArray x1 t) ≤ stype_size t"
      using comp_stotype_size_le by blast
    then have "stype_size (STArray x1 t) < stype_size (STArray x1 t)" by simp
    then show ?thesis by simp
  qed
next
  case (STMap x1 t)
  then have a1:"l1 = l2 ∨ (∃i. typeCon x1 i ∧ CompStoType t (STMap x1 t) (hash l1 i) l2)"
    unfolding CompStoType.simps by blast
  show "l1 = l2"
  proof (cases "l1 = l2")
    case True
    then show ?thesis by blast
  next
    case False
    then obtain i where idef: "typeCon x1 i ∧ CompStoType t (STMap x1 t) (hash l1 i) l2"
      using a1 by blast
    then have "stype_size (STMap x1 t) ≤ stype_size t"
      using comp_stotype_size_le by blast
    then have "stype_size (STMap x1 t) < stype_size (STMap x1 t)" by simp
    then show ?thesis by simp
  qed
next
  case (STValue x)
  then show ?case by simp
qed

definition compPointers :: "stack ⇒ (String.literal, type × denvalue) fmap ⇒ bool" where
  "compPointers st denval = (∀tp1 tp2 l1 l2 l1' l2' stl1 stl2.
    (type.Storage tp1, l1) |∈| fmran denval ∧ (type.Storage tp2, l2) |∈| fmran denval ∧
    ((l1 = Stackloc l1' ∧ accessStore l1' st = Some(KStoptr stl1)) ∨ (l1 = Storeloc stl1)) ∧
    ((l2 = Stackloc l2' ∧ accessStore l2' st = Some(KStoptr stl2)) ∨ (l2 = Storeloc stl2)) ⟶
    (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
     else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1
     else True))"
```

Overwriting one stack slot with a memory pointer preserves *compPointers*.

```
lemma compPointers_stack_update:
  assumes "compPointers sk den"
    and "∀loc. loc ≠ l ⟶ accessStore loc sk = accessStore loc sk'"
    and "accessStore l sk' = Some (KMemptr p)"
  shows "compPointers sk' den"
  unfolding compPointers_def
proof (intro allI impI)
  have sto: "accessStore x sk = Some (KStoptr q)" if "accessStore x sk' = Some (KStoptr q)" for x q
    using that assms(2,3) by (cases "x = l") auto
  fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
  assume "(type.Storage tp1, l1) |∈| fmran den ∧ (type.Storage tp2, l2) |∈| fmran den ∧
    (l1 = Stackloc l1' ∧ accessStore l1' sk' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
    (l2 = Stackloc l2' ∧ accessStore l2' sk' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
  then have "(type.Storage tp1, l1) |∈| fmran den ∧ (type.Storage tp2, l2) |∈| fmran den ∧
    (l1 = Stackloc l1' ∧ accessStore l1' sk = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
    (l2 = Stackloc l2' ∧ accessStore l2' sk = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
    using sto by blast
  then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
     else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
    using assms(1) unfolding compPointers_def by blast
qed
```

The pointer-compatibility condition of *compPointers* is symmetric.

```
lemma compPointers_if_sym:
```

```
  assumes "if TypedStoSubpref l1 l2 t2 then CompStoType t2 t1 l2 l1
           else if TypedStoSubpref l2 l1 t1 then CompStoType t1 t2 l1 l2 else True"
  shows "if TypedStoSubpref l2 l1 t1 then CompStoType t1 t2 l1 l2
         else if TypedStoSubpref l1 l2 t2 then CompStoType t2 t1 l2 l1 else True"
proof (cases "l1 = l2")
  case True
  then show ?thesis using assms CompStoType_sameLoc_sameType by (auto split: if_splits)
next
  case False
  then show ?thesis using assms typedStoSub_imps_negInv[OF False] by (auto split: if_splits)
qed


lemma TypedStoSubpref_child_imps_prnt:
  assumes "TypedStoSubpref (hash x4 i) p' t"
    and "CHR ''.'' ∉ set(String.explode i)"
    and "hash x4 i ≠ p'"
  shows "TypedStoSubpref x4 p' t"
  using assms
proof(induction t arbitrary:x4 i p')
  case (STArray x1 t)
  then show ?case using hash_injective typeConNoDots ShowLNatDot
    by (metis TypedStoSubpref.simps(2))
next
  case (STMap x1 t)
  then have "(∃i'. typeCon x1 i' ∧ TypedStoSubpref (hash x4 i) (hash p' i') t)" unfolding
TypedStoSubpref.simps by simp
  then obtain i' where i'def:"typeCon x1 i' ∧ TypedStoSubpref (hash x4 i) (hash p' i') t" by blast
  then have noDot:"CHR ''.'' ∉ set (literal.explode i')" using typeConNoDots[of x1 i'] i'def by blast
  then show ?case
  proof(cases "(hash x4 i) = (hash p' i')")
    case True
    then show ?thesis using hash_injective STMap True noDot
      by (metis TypedStoSubpref.simps(3))
  next
    case False
    then have "x4 = p' ∨ (∃i. typeCon x1 i ∧ TypedStoSubpref x4 (hash p' i) t)" using STMap.IH[of x4
i "(hash p' i')"] i'def noDot
      using STMap.prems(2) by blast
    then show ?thesis by simp
  qed
next
  case (STValue x)
  then show ?case unfolding TypedStoSubpref.simps by blast
qed

lemma TypedStoSubpref_specific_unreachanble_arry:
  assumes "destl ≠ srcl"
    and "¬ TypedStoSubpref srcl destl (STArray x t')"
    and "i < x"
  shows "¬ TypedStoSubpref (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t'"
proof

  assume *:"TypedStoSubpref (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t'"
  have **:"hash srcl (ShowL_nat i) ≠ hash destl (ShowL_nat i) " using assms
    by (metis hash_never_equal_prefix)
  then have " TypedStoSubpref srcl (hash destl (ShowL_nat i)) t'"
    using TypedStoSubpref_child_imps_prnt[OF * _ **] ShowLNatDot assms by auto
  then show False using assms by simp
qed

lemma NotRelatedPrnt_imps_notRelatedChild:
  assumes "¬ TypedStoSubpref x4 l''' t"
    and "¬ TypedStoSubpref l''' x4 struct"
```

```
    and "TypedStoSubpref l l''' t"
  shows "¬ TypedStoSubpref l x4 struct "
proof
  assume asm1:"TypedStoSubpref l x4 struct"
  show False using asm1 assms
  proof(induction struct arbitrary:x4)
    case (STArray x1 struct)
    have "l = x4 ∨ (∃i<x1. TypedStoSubpref l (hash x4 (ShowL_nat i)) struct)"
      using STArray.prems(1) unfolding TypedStoSubpref.simps by auto
    moreover have "l ≠ x4" using STArray.prems(2) assms(3) by auto
    ultimately obtain i where idef:"i<x1 ∧ TypedStoSubpref l (hash x4 (ShowL_nat i)) struct" by auto
    then have "CHR ''.'' ∉ set (literal.explode (ShowL_nat i))" using ShowLNatDot[of i] by blast
    then show ?case using idef
      by (metis STArray.IH STArray.prems(2,3) TypedStoSubpref_child_imps_prnt TypedStoSubpref_sameLoc
assms(3) TypedStoSubpref.simps(2))
  next
    case (STMap x1 struct)
    have "l = x4 ∨ (∃i. typeCon x1 i ∧ TypedStoSubpref l (hash x4 i) struct)"
      using STMap.prems(1) unfolding TypedStoSubpref.simps by blast
    moreover have "l ≠ x4" using STMap.prems(2) assms(3) by auto
    ultimately obtain i where idef:"typeCon x1 i ∧ TypedStoSubpref l (hash x4 i) struct" by blast
    then have iNoDot:" CHR ''.'' ∉ set (literal.explode i)" using typeConNoDots[of x1 i] by blast
    then show ?case using idef TypedStoSubpref_child_imps_prnt
      by (metis STMap.IH STMap.prems(2,3) TypedStoSubpref_sameLoc assms(3) TypedStoSubpref.simps(3))
  next
    case (STValue x)
    then show ?case by auto
  qed
qed


lemma CompStoType_same_type_same_depth:
  assumes "CompStoType tp'' t prnt l1"
    and "CompStoType tp'' t prnt' l2"
    and "location_depth prnt = location_depth prnt'"
  shows "location_depth l1 = location_depth l2" using assms
proof(induction tp'' arbitrary: prnt prnt')
  case (STArray x1 tp'')
  obtain i where idef: "t = STArray x1 tp'' ∧ prnt = l1
                      ∨ (i<x1 ∧ CompStoType tp'' t (hash prnt (ShowL_nat i)) l1)"
    using STArray.prems unfolding CompStoType.simps by blast
  obtain ii where iidef: "t = STArray x1 tp'' ∧ prnt' = l2
                       ∨ (ii<x1 ∧ CompStoType tp'' t (hash prnt' (ShowL_nat ii)) l2)"
    using STArray.prems unfolding CompStoType.simps by blast
  then show ?case
  proof(cases "t = STArray x1 tp''")
    case True
    then show ?thesis using idef iidef STArray.prems(3)
      using STArray.prems(1,2) comp_stotype_same_type_same_loc by blast
  next
    case False
    then have "CHR ''.'' ∉ set (String.explode (ShowL_nat ii)) ∧ CHR ''.'' ∉ set (String.explode
(ShowL_nat i))"
      using ShowLNatDot iidef idef by auto
    then have "location_depth (hash prnt' (ShowL_nat ii)) = location_depth (hash prnt' (ShowL_nat i))"
      using assms hash_adds_depth by metis
    then show ?thesis using STArray.IH False idef iidef
      by (metis STArray.prems(3) location_depth_hash_property)
  qed
next
  case (STMap x1 tp'')
  obtain i where idef: "t = STMap x1 tp'' ∧ prnt = l1 ∨
                       (typeCon x1 i ∧ CompStoType tp'' t (hash prnt i) l1)"
    using STMap.prems unfolding CompStoType.simps by blast
```

```
  obtain ii where iidef: "t = STMap x1 tp'' ∧ prnt' = l2
                        ∨ (typeCon x1 ii ∧ CompStoType tp'' t (hash prnt' ii) l2)"
    using STMap.prems unfolding CompStoType.simps by blast
  then show ?case
  proof(cases "t = STMap x1 tp''")
    case True
    then show ?thesis using idef iidef STMap.prems(3)
      using STMap.prems(1,2) comp_stotype_same_type_same_loc by blast
  next
    case False
    then have "CHR ''.'' ∉ set (String.explode ii) ∧ CHR ''.'' ∉ set (String.explode i)"
      using typeConNoDots iidef idef by auto
    then have "location_depth (hash prnt' ii) = location_depth (hash prnt' i)"
      using assms hash_adds_depth by metis
    then show ?thesis using STMap.IH False idef iidef
      by (metis STMap.prems(3) location_depth_hash_property)
  qed
next
  case (STValue x)
  then show ?case by auto
qed

lemma TypedStoSubpref_imp_same_depth:
  assumes "TypedStoSubpref l pa tp''"
    and "l ≠ pa"
  shows "∃a. location_depth (hash pa a) = location_depth l"
  using assms
proof(induction tp'' arbitrary:pa)
  case (STArray x1 tp'')
  then show ?case
    using TypedStoSubpref_b by blast
next
  case (STMap x1 tp'')
  then show ?case
    using TypedStoSubpref_b by blast
next
  case (STValue x)
  then show ?case by auto
qed

lemma same_depth_imp_neg_TypedStoSubpref:
  assumes "location_depth l = location_depth pa"
    and "l ≠ pa"
  shows "¬TypedStoSubpref l pa tp''" using assms TypedStoSubpref_imp_same_depth
  by (metis add_less_same_cancel1 less_add_same_cancel1 less_nat_zero_code less_one
location_depth_hash_property)


lemma CompStoType_unique_location:
  assumes "CompStoType struct tp1 (hash loc suffix1) p'"
    and "CompStoType struct tp2 (hash loc suffix2) p'"
    and "suffix1 ≠ suffix2"
    and "CHR ''.'' ∉ set(String.explode suffix1)"
    and "CHR ''.'' ∉ set(String.explode suffix2)"
  shows "False"
proof -
  have "hash loc suffix1 ≠ hash loc suffix2" using assms(3,4,5) hash_injective by blast
  moreover have "hash loc suffix1 = hash loc suffix2" using CompStoType_imps_subloc assms(1,2)
    by (metis CompStoType_imps_TypedStoSubpref NotRelatedPrnt_imps_notRelatedChild assms(4,5)
hash_adds_depth
      same_depth_imp_neg_TypedStoSubpref)
  ultimately show ?thesis by simp
qed
```

```
lemma sublocs_nonchanged_SCon:
  assumes "¬ TypedStoSubpref p' x4 struct"
    and "∀locs t''.
       locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
       accessStorage t'' locs sto = accessStorage t'' locs sto'"
    and "SCon struct x4 sto"
    and "¬ TypedStoSubpref x4 p' (STArray x t''')"
  shows "SCon struct x4 sto'" using assms
proof(induction struct arbitrary:x4)
  case (STArray x1 struct)
  have a0:"∀i<x1. SCon struct (hash x4 (ShowL_nat i)) sto"
    using STArray.prems(3) unfolding SCon.simps by simp
  moreover have "∀i<x1. SCon struct (hash x4 (ShowL_nat i)) sto'"
  proof intros
    fix i assume *:"i<x1"
    have "¬ TypedStoSubpref p' (hash x4 (ShowL_nat i)) struct"
      using STArray.prems(1) * unfolding TypedStoSubpref.simps by auto
    moreover have "SCon struct (hash x4 (ShowL_nat i)) sto"
      using calculation * a0 by auto
    moreover have "¬ TypedStoSubpref (hash x4 (ShowL_nat i)) p' (STArray x t''')"
    proof
      assume asm:"TypedStoSubpref (hash x4 (ShowL_nat i)) p' (STArray x t''')"
      have "hash x4 (ShowL_nat i) ≠ p'" using TypedStoSubpref_sameLoc asm
        using calculation(1) by blast
      then have "TypedStoSubpref x4 p' (STArray x t''')"
        using TypedStoSubpref_child_imps_prnt[OF asm _ ] ShowLNatDot by blast
      then show False using STArray.prems(4) by contradiction
    qed
    ultimately show "SCon struct (hash x4 (ShowL_nat i)) sto'"
      using STArray.IH[of "hash x4 (ShowL_nat i)"] STArray.prems(2) by blast
  qed
  ultimately show ?case unfolding SCon.simps by simp
next
  case (STMap x1 struct)
  have t:"∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto"
    using STMap.prems(3) unfolding SCon.simps by simp
  have "∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto'"
  proof intros
    fix i assume *:"typeCon x1 i"
    moreover have "¬ TypedStoSubpref p' (hash x4 i) struct"
      using STMap.prems(1) calculation by auto
    moreover have "∀locs t''.
       locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
       accessStorage t'' locs sto = accessStorage t'' locs sto'"
      using STMap.prems(2) * by simp
    moreover have "SCon struct (hash x4 i) sto" using t * by blast
    moreover have "¬ TypedStoSubpref (hash x4 i) p' (STArray x t''')"
      using STMap.prems(4) calculation(2) TypedStoSubpref_child_imps_prnt
      using "*" TypedStoSubpref_sameLoc typeConNoDots by blast
    ultimately show "SCon struct (hash x4 i) sto'" using STMap.IH[of "(hash x4 i)"] by blast
  qed
  then show ?case unfolding SCon.simps by blast
next
  case (STValue x')
  have "typeCon x' (accessStorage x' x4 sto)" using STValue(3) unfolding SCon.simps by blast
  have "x4 ≠ p'" using STValue(1) by simp
  then have "(accessStorage x' x4 sto) = (accessStorage x' x4 sto')" using STValue(2,4) by blast
  then show ?case using ‹typeCon x' (accessStorage x' x4 sto)› by auto
qed


lemma singleLocChanged_nonchanged_SCon:
  assumes "∀l. l ≠ locationChanged ⟶ sto' $$ l = sto $$ l"
    and "x4 ≠ locationChanged"
```

```
    and "SCon struct x4 sto"
    and "¬ TypedStoSubpref l'' x4 struct"
    and "¬ TypedStoSubpref x4 l'' t''"
    and " TypedStoSubpref locationChanged l'' t'' ∧ CompStoType t'' (STValue t') l'' locationChanged"
  shows " SCon struct x4 sto'" using assms(2,3,4,5,6)
proof(induction struct arbitrary:x4)
  case (STArray x1 struct)
  have asm2:"∀ i<x1. SCon struct (hash x4 (ShowL_nat i)) sto" using STArray unfolding SCon.simps by
auto
  have "∀ i<x1. SCon struct (hash x4 (ShowL_nat i)) sto'"
  proof intros
    fix i assume asm3:"i<x1"
    then have "SCon struct (hash x4 (ShowL_nat i)) sto" using asm2 by blast
    moreover have "(hash x4 (ShowL_nat i)) ≠ locationChanged" using TypedStoSubpref_child_imps_prnt
STArray.prems
      using asm3 ShowLNatDot
      using TypedStoSubpref_sameLoc TypedStoSubpref.simps(2) by blast
    moreover have "¬ TypedStoSubpref l'' (hash x4 (ShowL_nat i)) struct" using STArray.prems asm3 by
simp
    moreover have "(hash x4 (ShowL_nat i)) ≠ l''"
      using TypedStoSubpref_sameLoc calculation(3) by auto
    moreover have "¬ TypedStoSubpref (hash x4 (ShowL_nat i)) l'' t'' " using STArray.prems
TypedStoSubpref_child_imps_prnt[of x4] asm3 ShowLNatDot
      using calculation(4) by blast
    ultimately show "SCon struct (hash x4 (ShowL_nat i)) sto'" using STArray(1) STArray.prems(5)
      using assms(1) by auto
  qed
  then show ?case by simp
next
  case (STMap x1 struct)
  have asm2:"∀ i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto" using STMap by auto
  have "∀ i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto'"
  proof intros
    fix i assume asm3:"typeCon x1 i"
    then have "SCon struct (hash x4 i) sto" using asm2 by blast
    moreover have "hash x4 i ≠ locationChanged" using TypedStoSubpref_child_imps_prnt STMap
      using asm3 typeConNoDots
      by (metis TypedStoSubpref_sameLoc TypedStoSubpref.simps(3))
    moreover have "¬ TypedStoSubpref l'' (hash x4 i) struct" using STMap asm3 by simp
    moreover have "hash x4 i ≠ l''"
      using TypedStoSubpref_sameLoc calculation(3) by auto
    moreover have "¬ TypedStoSubpref (hash x4 i) l'' t'' " using STMap.prems
TypedStoSubpref_child_imps_prnt[of x4 i l'' t''] asm3 typeConNoDots
      using calculation(4) by blast
    ultimately show "SCon struct (hash x4 i) sto'" using STMap(1)[of "(hash x4 i)"] STMap.prems(5)
      using assms(1) by auto
  qed
  then show ?case unfolding SCon.simps by simp
next
  case (STValue x)
  then show ?case using assms(1)
    by (simp add: accessStorage_def)
qed

lemma SCon_imps_sublocs:
  assumes "CompStoType t' struct l x2"
    and "SCon t' l sto'"
  shows "SCon struct x2 sto'"
  using assms
proof(induction t' arbitrary: l)
  case (STArray x1 t''')
  have c:"struct = STArray x1 t''' ∧ l = x2 ∨ (∃ i<x1. CompStoType t''' struct (hash l (ShowL_nat i))
x2)"
    using STArray.prems(1) unfolding CompStoType.simps by simp
```

```
  then show ?case
  proof(cases "l = x2")
    case True
    then show ?thesis using c STArray
      by auto
  next
    case False
    then show ?thesis using c STArray by auto
  qed
next
  case (STMap x1 t''')
  have c:"struct = STMap x1 t''' ∧ l = x2 ∨ (∃i. typeCon x1 i ∧ CompStoType t''' struct (hash l i)
x2)"
    using STMap.prems(1) unfolding CompStoType.simps by simp
  then show ?case
  proof(cases "l = x2")
    case True
    then show ?thesis
      using CompStoType_sameLoc_sameType STMap.prems(1,2) by blast
  next
    case f1:False
    then obtain i where idef: "(typeCon x1 i ∧ CompStoType t''' struct (hash l i) x2)"
      using c by blast
    then show ?thesis using STMap.IH
      using STMap.prems(2) by auto
  qed
next
  case (STValue x)
  then show ?case by auto
qed


lemma TypedStoSubpref_shared_parent_related:
  assumes "TypedStoSubpref lc l'' t''"
    and "TypedStoSubpref lc x4 struct"
  shows "∃pref. hash l'' pref = x4 ∨ hash x4 pref = l'' ∨ l'' =x4 "
proof -
  obtain pre1 where a10: "hash l'' pre1 = lc ∨ lc = l''" using TypedStoSubpref_b assms by blast
  obtain pre2 where a20:"hash x4 pre2 = lc ∨ lc = x4" using assms TypedStoSubpref_b by blast
  then show ?thesis
  proof (cases "pre1 = pre2")
    case True
    then show ?thesis using a10 a20 hash_never_equal_prefix by auto
  next
    case False
    then show ?thesis by (metis LSubPrefL2_def a10 a20 Mutual_NonSub_SpecificNonSub)
  qed
qed


lemma subLocs:
  assumes "SCon struct x4 sto"
    and "∀l struct. ¬TypedStoSubpref locationChanged l struct ⟶ sto' $$ l = sto $$ l"
    and "(∀i'. (hash x4 i') ≠ locationChanged)"
    and "x4 ≠ locationChanged"
  shows " SCon struct x4 sto'" using assms(2) assms(1) assms(3) assms(4)
proof(induction struct arbitrary: x4)
  case (STArray x1 struct)
  have "(∀i<x1. SCon struct (hash x4 (ShowL_nat i)) sto')"
  proof(intros)
    fix i assume iless:"i<x1"
    then have "SCon struct (hash x4 (ShowL_nat i)) sto" using STArray(3) by simp
    moreover have "∀l struct. ¬ TypedStoSubpref locationChanged l struct ⟶ sto' $$ l = sto $$ l"
using STArray by auto
```

```
    moreover have " ∀i'. hash (hash x4 (ShowL_nat i)) i' ≠ locationChanged" using STArray by (simp
add: hash_suffixes_associative)
    moreover have "hash x4 (ShowL_nat i) ≠ locationChanged" using STArray by blast
    ultimately show "SCon struct (hash x4 (ShowL_nat i)) sto' " using STArray(1)[of "(hash x4 (ShowL_nat
i))"] by blast
  qed
  then show ?case using SCon.simps(2)[of x1 struct x4 sto'] by auto
next
  case (STMap x1 struct)
  have "(∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto')"
  proof intros
    fix i assume *:"typeCon x1 i"
    then have "SCon struct (hash x4 i) sto" using STMap(3) by simp
    moreover have "∀l struct. ¬ TypedStoSubpref locationChanged l struct ⟶ sto' $$ l = sto $$ l"
using STMap by auto
    moreover have " ∀i'. hash (hash x4 i) i' ≠ locationChanged" using STMap by (simp add:
hash_suffixes_associative)
    moreover have "hash x4 i ≠ locationChanged" using STMap by blast
    ultimately show "SCon struct (hash x4 i) sto'" using STMap(1)[of "(hash x4 i)"] by blast
  qed
  then show ?case using SCon.simps(3)[of x1 struct x4] by simp
next
  case (STValue x)
  have "typeCon x (accessStorage x x4 sto')" unfolding accessStorage_def using STValue(1) STValue(2)
STValue(4)
    by (metis accessStorage_def SCon.simps(1) TypedStoSubpref.simps(1))
  then show ?case using SCon.simps(1)[of x x4] by auto
qed


lemma stvalueLocationsInduct:
  assumes "CompStoType struct (STValue t') x4 locationChanged"
    and " SCon struct x4 sto"
    and "SCon (STValue t') locationChanged sto'"
    and "hash x4 i'' = locationChanged"
    and "∀l struct. ¬TypedStoSubpref locationChanged l struct ⟶ sto' $$ l = sto $$ l"
    and "∀l. l ≠ locationChanged ⟶ sto' $$ l = sto $$ l"
  shows "SCon struct x4 sto'" using assms(1) assms(2) assms(3) assms(4) assms(5)
proof(induction struct arbitrary:x4 i'')
  case (STArray x1 struct)
  then have c5:" ((∃i<x1. CompStoType struct (STValue t') (hash x4 (ShowL_nat i)) locationChanged))"
    using CompStoType.simps(2)[of x1 struct "(STValue t')" x4 locationChanged] by auto
  have "(∀i<x1. SCon struct (hash x4 (ShowL_nat i)) sto')"
  proof intros
    fix i assume iless:"i<x1"
    show "SCon struct (hash x4 (ShowL_nat i)) sto' "
    proof(cases "(ShowL_nat i) = i'')
      case True
      then have "locationChanged = (hash x4 (ShowL_nat i))" by (simp add: STArray.prems)
      moreover have "struct = (STValue t')"
        using c5 True iless CompStoType_sameLoc_sameType CompStoType_imps_subloc STArray.prems(4)
ShowLNatDot
        by (metis LSubPrefL2_def typeCon_no_sublocation_prefix checkAddress_def typeCon.simps(4))
      ultimately have "SCon struct locationChanged sto'"
        using STArray by auto
      then show ?thesis using STArray True by simp
    next
      case False
      then have c10:"SCon struct (hash x4 (ShowL_nat i)) sto" using STArray iless by simp
      have c20:"hash x4 (ShowL_nat i) ≠ locationChanged" using False STArray.prems
hash_never_equal_suffix by auto
      have c30:" SCon (STValue t') locationChanged sto'" using STArray(4) by auto
      then show ?thesis
      proof(cases " CompStoType struct (STValue t') (hash x4 (ShowL_nat i)) locationChanged")
```

```
        case True
        then show ?thesis using STArray(1) c10 c20 c30 False
          by (meson CompStoType_imps_subloc STArray.prems(3) STArray(6))
      next
        case False
        then have "(∀i'. (hash (hash x4 (ShowL_nat i)) i') ≠ locationChanged)"
          using False by (metis (full_types) CompStoType_imps_subloc c5 hash_suffixes_associative
hashesInts)
        then show ?thesis using subLocs[of struct "(hash x4 (ShowL_nat i))" _ _] c10 STArray(6) c20
False by auto
      qed
    qed
  qed
  then show ?case using SCon.simps(2)[of x1 struct x4 "sto'"] by simp
next
  case (STMap x1 struct)
  then have c5:" ((∃i. typeCon x1 i ∧ CompStoType struct (STValue t') (hash x4 i) locationChanged))"
    using CompStoType.simps(3)[of x1 struct "(STValue t')" x4 locationChanged] by simp
  then obtain i''' where i'''def:" typeCon x1 i''' ∧ CompStoType struct (STValue t') (hash x4 i''')
locationChanged" by auto
  have "(∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto')"
  proof intros
    fix i assume iless:"typeCon x1 i"
    show "SCon struct (hash x4 i) sto' "
    proof(cases "i = i'''")
      case t9:True
      then have locsSame:"locationChanged = (hash x4 i)"
        by (simp add: STMap.prems(4))
      then have "struct = (STValue t')"
        using c5 t9 iless locsSame CompStoType_sameLoc_sameType[of struct "STValue t'" "hash x4 i"]
        by (metis CompStoType_imps_subloc LSubPrefL2_def typeCon_no_sublocation_prefix)
      then have "SCon struct locationChanged sto'"
        using STMap(4) by auto
      then show ?thesis using STMap t9 by force
    next
      case f3:False
      then have c10:"SCon struct (hash x4 i) sto" using STMap iless by simp
      have c20:"hash x4 i ≠ locationChanged" using f3 STMap.prems(4) hash_never_equal_suffix by auto
      have c30:"SCon (STValue t') locationChanged sto'" using STMap(4) by auto
      then show ?thesis
      proof(cases " CompStoType struct (STValue t') (hash x4 i) locationChanged")
        case True
        then show ?thesis using STMap(1) c10 c20 c30 f3

          by (meson CompStoType_imps_subloc STMap(6))
      next
        case False
        then have "(∀i'. (hash (hash x4 (i)) i') ≠ locationChanged)" using c5
Mutual_NonSub_SpecificNonSub
          using False f3 iless typeCon_no_sublocation_prefix i'''def c10 CompStoType_imps_subloc
          by (metis LSubPrefL2_def)
        then show ?thesis using subLocs[of struct "(hash x4 i)"] c10 STMap(6) c20 False by auto
      qed
    qed
  qed
  then show ?case using SCon.simps(3)[of x1 struct x4 "sto'"] by simp
next
  case (STValue x)
  then show ?case using SCon.simps(1)[of x x4 "sto'"] assms(6) by simp
qed

lemma SCon_value_write_imps_SCon:
  assumes comp: "CompStoType struct (STValue t') x4 locationChanged"
    and scOld: "SCon struct x4 sto"
```

```
    and scNew: "SCon (STValue t') locationChanged sto'"
    and nonLocChanged: "∀l. l ≠ locationChanged ⟶ sto' $$ l = sto $$ l"
  shows "SCon struct x4 sto'"
proof (cases "x4 = locationChanged")
  case True
  then have "struct = STValue t'"
    using comp by (simp add: CompStoType_sameLoc_sameType)
  then show ?thesis using True scNew by simp
next
  case False
  then obtain i'' where iDef: "hash x4 i'' = locationChanged"
    using comp CompStoType_imps_subloc by blast
  have nonSubChanged:
    "∀l struct'. ¬ TypedStoSubpref locationChanged l struct' ⟶ sto' $$ l = sto $$ l"
    by (metis TypedStoSubpref_sameLoc nonLocChanged)
  show ?thesis
    using stvalueLocationsInduct[OF comp scOld scNew iDef nonSubChanged nonLocChanged] by blast
qed

lemma CompStoType_subloc_type_transfer:
  assumes "CompStoType t'' (STValue t') l'' locationChanged "
    and" CompStoType t'' struct l'' x4 "
    and "TypedStoSubpref locationChanged x4 struct"
  shows " CompStoType struct (STValue t') x4 locationChanged" using assms
proof(induction t'' arbitrary:l'')
  case (STArray x1 t'')
  then show ?case
  proof(cases "l'' = locationChanged")
    case True
    then show ?thesis using STArray using CompStoType_sameLoc_sameType by blast
  next
    case f1:False
    then show ?thesis
    proof(cases "l'' = x4")
      case True
      then show ?thesis using STArray f1 using CompStoType_sameLoc_sameType by blast
    next
      case False
      then obtain i1 where a10:"(i1<x1 ∧ CompStoType t'' (STValue t') (hash l'' (ShowL_nat i1))
locationChanged)"
        using STArray(2) CompStoType.simps(2)[of x1 t'' "STValue t'" l'' locationChanged] using f1 by
blast
      then obtain i2 where a20:" (i2<x1 ∧ CompStoType t'' struct (hash l'' (ShowL_nat i2)) x4)"
        using STArray(3) CompStoType.simps(2)[of x1 t'' "struct" l'' x4] using False by blast
      then have a30:"i1 = i2" using a10 a20 STArray
      proof -
        obtain ll :: "String.literal ⇒ String.literal ⇒ String.literal" where
          f1: "∀l la s sa. hash l (ll la l) = la ∨ ¬ CompStoType s sa l la ∨ l = la"
          by (metis (no_types) CompStoType_imps_subloc)
        obtain lla :: "String.literal ⇒ String.literal ⇒ String.literal" where
          f2: "locationChanged = hash x4 (lla x4 locationChanged) ∨ x4 = locationChanged"
          using TypedStoSubpref_b assms(3) by fastforce
        then have f3: "∀l. hash x4 (hash (lla x4 locationChanged) l) = hash locationChanged l ∨ x4 =
locationChanged"
          by (metis hash_suffixes_associative)
        have f4: "x4 = hash (hash l'' (ShowL_nat i2)) (ll x4 (hash l'' (ShowL_nat i2))) ∨ x4 = hash l''
(ShowL_nat i2)"
          using f1 by (metis (full_types) a20)
        have f5: "locationChanged = hash (hash l'' (ShowL_nat i1)) (ll locationChanged (hash l''
            (ShowL_nat i1))) ∨ locationChanged = hash l'' (ShowL_nat i1)"
          using f1 by (metis a10)
        have f6: "∀l. hash x4 l = hash (hash l'' (ShowL_nat i2)) (hash (ll x4 (hash l''
            (ShowL_nat i2))) l) ∨ x4 = hash l'' (ShowL_nat i2)"
          using f4 by (metis hash_suffixes_associative)
```

```
        have f7: "∀l n. locationChanged ≠ hash (hash l'' (ShowL_nat n)) l ∨ i1 = n ∨ locationChanged =
hash l'' (ShowL_nat i1)"
          using f5 by (meson hashesInts)
        { assume "∃l la. hash locationChanged l = hash locationChanged la ∧ i2 ≠ i1"
          { assume "∃l la. hash locationChanged la = hash locationChanged l ∧ x4 ≠ locationChanged"
        moreover
        { assume "∃l la. hash locationChanged la = hash x4 l ∧ i2 ≠ i1 ∧ x4 ≠ locationChanged"
        moreover
        { assume "∃l la. hash locationChanged la = hash locationChanged l ∧ x4 ≠ hash l'' (ShowL_nat
i2) ∧ x4 ≠ locationChanged"
        then have "locationChanged ≠ hash l'' (ShowL_nat i1) ∧ x4 ≠ locationChanged ∨ i1 = i2"
          using f6 f3 by (metis (no_types) hashesInts) }
        ultimately have "x4 ≠ hash l'' (ShowL_nat i2) ∧ locationChanged ≠ hash l'' (ShowL_nat i1) ∨ i1
= i2"
          using f7 f2 by (metis (no_types) hashesInts) }
        ultimately have "x4 ≠ hash l'' (ShowL_nat i2) ∧ locationChanged ≠ hash l'' (ShowL_nat i1) ∨ i1
= i2"
          using f3 by (metis (full_types)) }
        then have ?thesis
          using f5 f4 f2 by (metis hash_suffixes_associative hashesInts) }
        then show ?thesis
          by blast
      qed

      then show ?thesis
      proof(cases "x4 = locationChanged")
        case True
        then show ?thesis
          using STArray.IH a10 a20 a30 assms(3) by auto
      next
        case False
        then show ?thesis
          using STArray.IH a10 a20 a30 assms(3) by auto
      qed
    qed
  qed
next
  case (STMap x1 t'')
  then show ?case
  proof(cases "l'' = locationChanged")
    case True
    then show ?thesis using STMap using CompStoType_sameLoc_sameType by blast
  next
    case f1:False
    then show ?thesis
    proof(cases "l'' = x4")
      case True
      then show ?thesis using STMap f1 using CompStoType_sameLoc_sameType by blast
    next
      case False
      then obtain i1 where a10:"(typeCon x1 i1 ∧ CompStoType t'' (STValue t') (hash l'' i1)
locationChanged)"
        using STMap(2) CompStoType.simps(3)[of x1 t'' "STValue t'" l'' locationChanged] using f1 by
blast
      then obtain i2 where a20:" (typeCon x1 i2 ∧ CompStoType t'' struct (hash l'' i2) x4)"
        using STMap(3) CompStoType.simps(3) using False by blast
      have a30:"CHR ''.'' ∉ set(String.explode i1)" using a10 using typeConNoDots by auto
      have a40:"CHR ''.'' ∉ set(String.explode i2)" using a20 using typeConNoDots by auto

      then have a30:"i1 = i2" using a10 a20 STMap
      proof -
        have "LSubPrefL2 locationChanged (hash l'' i1)"
          using CompStoType_imps_subloc a10 unfolding LSubPrefL2_def by metis
        moreover have "LSubPrefL2 x4 (hash l'' i2)"
```

```
        using CompStoType_imps_subloc a20 unfolding LSubPrefL2_def by metis
      moreover have "LSubPrefL2 locationChanged x4"
        using STMap.prems(3) TypedStoSubpref_b unfolding LSubPrefL2_def by metis
      ultimately show ?thesis
        using LSubPrefL2_hash_unique LSubPrefL2_trans a30 a40 by blast
    qed

    then show ?thesis
    proof(cases "x4 = locationChanged")
      case True
      then show ?thesis
        using STMap.IH a10 a20 a30 assms(3) by auto
    next
      case False
      then show ?thesis
        using STMap.IH a10 a20 a30 assms(3) by auto
    qed
  qed
qed
next
  case (STValue x)
  then show ?case by simp
qed

lemma CompStoType_sameLocs_sameType:
  assumes "CompStoType t t' l''' x4"
    and "CompStoType t struct l''' x4"
  shows "struct = t'" using assms
proof(induction t arbitrary:l''')
  case (STArray x1 t)
  have asm1:"t' = STArray x1 t ∧ l''' = x4 ∨ (∃i<x1. CompStoType t t' (hash l''' (ShowL_nat i)) x4)"
    using STArray.prems unfolding CompStoType.simps by simp
  have asm2:"struct = STArray x1 t ∧ l''' = x4 ∨ (∃i<x1. CompStoType t struct (hash l''' (ShowL_nat i))
x4)"
    using STArray.prems unfolding CompStoType.simps by simp
  then show ?case
  proof(cases "l''' = x4")
    case True
    then have "t' = STArray x1 t " using asm1 asm2
      using CompStoType_sameLoc_sameType STArray.prems assms by blast
    moreover have "struct = STArray x1 t " using asm1 asm2
      using CompStoType_sameLoc_sameType STArray.prems True by blast
    ultimately show ?thesis by blast
  next
    case False
    then have "(∃i<x1. CompStoType t t' (hash l''' (ShowL_nat i)) x4)" using asm1 by simp
    moreover have "(∃i<x1. CompStoType t struct (hash l''' (ShowL_nat i)) x4)" using asm2 False by
simp
    ultimately show ?thesis using STArray.IH
      using CompStoType_unique_location typeConNoDots
      by (metis ShowLNatDot)
  qed
next
  case (STMap x1 t)
  have asm1:"t' = STMap x1 t ∧ l''' = x4 ∨ (∃i. typeCon x1 i ∧ CompStoType t t' (hash l''' i) x4)"
    using STMap.prems unfolding CompStoType.simps by simp
  have asm2:"struct = STMap x1 t ∧ l''' = x4 ∨ (∃i. typeCon x1 i ∧ CompStoType t struct (hash l''' i)
x4)"
    using STMap.prems unfolding CompStoType.simps by simp
  then show ?case
  proof(cases "l''' = x4")
    case True
    then have "t' = STMap x1 t" using asm1 asm2
      using CompStoType_sameLoc_sameType STMap.prems assms by blast
```

```
    moreover have "struct = STMap x1 t " using asm1 asm2
      using CompStoType_sameLoc_sameType STMap.prems True by blast
    ultimately show ?thesis by blast
  next
    case False
    then have "(∃i. typeCon x1 i ∧ CompStoType t t' (hash l''' i) x4)" using asm1 by simp
    moreover have "(∃i. typeCon x1 i ∧ CompStoType t struct (hash l''' i) x4)" using asm2 False by
simp
    ultimately show ?thesis using STMap.IH
      using CompStoType_unique_location typeConNoDots by blast

  qed
next
  case (STValue x)
  then show ?case by simp
qed


lemma CompStoType_trns:
  assumes "CompStoType struct t'' x4 l'''"
    and "CompStoType t'' t' l'' locationChanged"
  shows "CompStoType struct t' x4 locationChanged" using assms
proof(induction struct arbitrary:x4)
  case (STArray x1 struct)
  show ?case

  proof(cases "x4 = l'''")
    case True
    then have "t'' = STArray x1 struct" using STArray.prems CompStoType_sameLoc_sameType by blast
    then have "CompStoType (STArray x1 struct) t' x4 locationChanged" using STArray.prems True by simp
    then show ?thesis by simp
  next
    case False
    then have "(∃i<x1. CompStoType struct t'' (hash x4 (ShowL_nat i)) l'')"
      using STArray.prems CompStoType.simps(2)[of x1 struct t'' x4 l''] by simp
    then obtain i where a10:"i<x1 ∧ CompStoType struct t'' (hash x4 (ShowL_nat i)) l''" by auto
    then have "CompStoType struct t' (hash x4 (ShowL_nat i)) locationChanged" using STArray
STArray.prems by simp
    then have " (∃i<x1. CompStoType struct t' (hash x4 (ShowL_nat i)) locationChanged)" using a10 by
auto
    then show ?thesis by simp
  qed

next
  case (STMap x1 struct)
  show ?case

  proof(cases "x4 = l'''")
    case True
    then have "t'' = STMap x1 struct" using STMap.prems CompStoType_sameLoc_sameType by blast
    then have "CompStoType (STMap x1 struct) t' x4 locationChanged" using STMap.prems True by simp
    then show ?thesis by simp
  next
    case False
    then have "(∃i. typeCon x1 i ∧ CompStoType struct t'' (hash x4 i) l'')"
      using STMap.prems CompStoType.simps(3)[of x1 struct t'' x4 l''] by simp
    then obtain i where a10:"typeCon x1 i ∧ CompStoType struct t'' (hash x4 i) l''" by auto
    then have "CompStoType struct t' (hash x4 i) locationChanged" using STMap STMap.prems by simp
    then have " (∃i. typeCon x1 i ∧ CompStoType struct t' (hash x4 i) locationChanged)" using a10 by
auto
    then show ?thesis by simp
  qed

next
```

```
  case (STValue x)
  then show ?case by auto
qed

lemma NotReachablePrnt_imps_notReachableChild:
  assumes "¬ TypedStoSubpref x4 l''' t"
    and "¬ TypedStoSubpref l''' x4 struct"
    and "CompStoType t t' l''' l"
  shows "¬ TypedStoSubpref x4 l t'"
proof
  assume asm1:"TypedStoSubpref x4 l t'"
  show False using asm1 assms
  proof(induction t' arbitrary:l l''')
    case (STArray x1 struct)
    have asm2:"x4 = l ∨ (∃ i<x1. TypedStoSubpref x4 (hash l (ShowL_nat i)) struct)"
      using STArray.prems(1) unfolding TypedStoSubpref.simps by blast
    then show ?case
    proof(cases "x4 = l")
      case True
      then show ?thesis
        using CompStoType_imps_TypedStoSubpref STArray.prems by auto
    next
      case False
      then obtain i where idef:"(i<x1 ∧ TypedStoSubpref x4 (hash l (ShowL_nat i)) struct)" using asm2
by auto
      then have asm4:"CompStoType (STArray x1 struct) struct l (hash l (ShowL_nat i))"
        using CompStoType_sameLocNdTyp by auto
      then have iNoDot:" CHR ''.'' ∉ set (literal.explode (ShowL_nat i))" using idef ShowLNatDot by
blast

      then show ?thesis
      proof(cases "l = l''''")
        case True
        then show ?thesis
          using CompStoType_sameLoc_sameType STArray.prems by blast
      next
        case False
        then have "CompStoType t struct l''' (hash l (ShowL_nat i)) " using idef STArray.prems asm4
          using CompStoType_trns by blast
        then show ?thesis using STArray.IH STArray.prems(2,3) idef by blast
      qed
    qed
  next
    case (STMap x1 struct)
    have asm2:"x4 = l ∨ (∃ i. typeCon x1 i ∧ TypedStoSubpref x4 (hash l i) struct)"
      using STMap.prems(1) unfolding TypedStoSubpref.simps by blast
    then show ?case
    proof(cases "x4 = l")
      case True
      then show ?thesis
        using CompStoType_imps_TypedStoSubpref STMap.prems(4) assms(1)
        using STMap.prems(2) by auto
    next
      case False
      then obtain i where idef:"(typeCon x1 i ∧ TypedStoSubpref x4 (hash l i) struct)" using asm2 by
auto
      then have asm4:"CompStoType (STMap x1 struct) struct l (hash l i)"
        using CompStoType_sameLocNdTyp by auto
      then have iNoDot:" CHR ''.'' ∉ set (literal.explode i)" using idef typeConNoDots[of x1 i] by
blast

      then show ?thesis
      proof(cases "l = l''''")
        case True
```

```
      then show ?thesis
        using CompStoType_sameLoc_sameType STMap.prems(1,4)
        using STMap.prems(2) by blast
    next
      case False
      then have "CompStoType t struct l''' (hash l i) " using idef STMap.prems(4) asm4
        using CompStoType_trns by blast
      then show ?thesis using STMap.IH STMap.prems(2,3) idef by blast
    qed
  qed
next
  case (STValue x)
  then have "x4 = l" unfolding TypedStoSubpref.simps by simp
  then have "CompStoType t (STValue x) l''' x4" using STValue by blast
  then have "TypedStoSubpref x4 l''' t"
    using CompStoType_imps_TypedStoSubpref[of t _ l''' x4] by simp
  then show ?case using STValue by simp
qed
qed

lemma SCon_noDots:
  assumes "SCon tp (hash destl i) v'"
    and "∀i' t. accessStorage t (hash (hash destl i) i') v' = accessStorage t (hash (hash destl i) i')
v''"
    and " ∀t. accessStorage t (hash destl i) v' = accessStorage t (hash destl i) v''"
    and "CHR ''.'' ∉ set(String.explode i)"
  shows "SCon tp (hash destl i) v''" using assms
proof (induction tp arbitrary: destl i)
  case (STArray x1 tp)

  have " ∀i'<x1. SCon tp (hash (hash destl i) (ShowL_nat i')) v''"
  proof intros
    fix i' assume *:"i'<x1"
    then have "SCon tp (hash (hash destl i) (ShowL_nat i')) v'" using STArray.prems by simp
    moreover have "∀i'a t. accessStorage t (hash (hash (hash destl i) (ShowL_nat i')) i'a) v' =
accessStorage t
        (hash (hash (hash destl i) (ShowL_nat i')) i'a) v''" using STArray.prems
      by (simp add: hash_suffixes_associative)
    moreover have "CHR ''.'' ∉ set (literal.explode (ShowL_nat i'))" using ShowLNatDot by simp
    moreover have "∀t. accessStorage t (hash (hash destl i) (ShowL_nat i')) v' = accessStorage t (hash
(hash
        destl i) (ShowL_nat i')) v''" using STArray.prems by simp
    ultimately show "SCon tp (hash (hash destl i) (ShowL_nat i')) v''" using STArray.IH[of "(hash destl
i)" "(ShowL_nat i')"] by blast
  qed
  then show ?case unfolding SCon.simps by simp
next
  case (STMap x1 tp)
  have "∀i'. typeCon x1 i' ⟶ SCon tp (hash (hash destl i) i') v''"
  proof intros
    fix i' assume *:"typeCon x1 i'"
    then have "CHR ''.'' ∉ set (literal.explode i')" using typeConNoDots by simp
    moreover have "SCon tp (hash (hash destl i) i') v'" using * STMap.prems by simp
    moreover have "∀i'a t. accessStorage t (hash (hash (hash destl i) i') i'a) v' = accessStorage t
(hash (hash (hash destl i) i') i'a) v''" using STMap.prems
      by (simp add: hash_suffixes_associative)
    moreover have "∀t. accessStorage t (hash (hash destl i) i') v' = accessStorage t (hash (hash
destl i) i') v''" using STMap.prems by simp
    ultimately show "SCon tp (hash (hash destl i) i') v''" using STMap.IH by blast
  qed
  then show ?case unfolding SCon.simps by simp
next
  case (STValue x)
  then show ?case by auto
```

qed

lemma *Sto_divergence_imps_notsubloc:*
shows *"¬TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) destl (STArray x tp)"*
proof
assume *"TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) destl (STArray x tp)"*
then have **:"hash destl (ShowL$_{nat}$ x) = destl ∨ (∃ i<x. TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash destl (ShowL$_{nat}$ i)) tp)"*
unfolding *TypedStoSubpref.simps* **by** *simp*
then have *"hash destl (ShowL$_{nat}$ x) ≠ destl"* **using** *ShowLNatDot*
using *hash_inequality* **by** *auto*
then obtain *i* **where** *asm4:"(i<x ∧ TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash destl (ShowL$_{nat}$ i)) tp)"*
using *** **by** *blast*
then show *False*
proof *(cases tp)*
case *(STArray x1 tp)*
have **:" i < x ∧ (hash destl (ShowL$_{nat}$ x) = hash destl (ShowL$_{nat}$ i) ∨ (∃ ia. ia <x1 ∧ TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash (hash destl (ShowL$_{nat}$ i)) (ShowL$_{nat}$ ia)) tp))"*
using *asm4 STArray* **unfolding** *TypedStoSubpref.simps* **by** *simp*
then have *"hash destl (ShowL$_{nat}$ x) ≠ hash destl (ShowL$_{nat}$ i)"* **using** *hash_injective ShowLNatDot Read_Show_nat'_id readLintNotEqual ** **by** *metis*
then obtain *ia* **where** *" ia <x1 ∧ TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash (hash destl (ShowL$_{nat}$ i)) (ShowL$_{nat}$ ia)) tp"* **using** *** **by** *auto*
then have *a3:"TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash (hash destl (ShowL$_{nat}$ i)) (ShowL$_{nat}$ ia)) tp"*
by *(simp add: hash_suffixes_associative)*
then have *" (hash (ShowL$_{nat}$ i) (ShowL$_{nat}$ ia)) ≠ (ShowL$_{nat}$ x)"* **using** *hash_def ShowLNatDot*
by *(metis subPrefCannotBeInt)*
then show *?thesis* **using** *a3 hash_injective ShowLNatDot TypedStoSubpref_child_imps_prnt TypedStoSubpref_hashes hash_never_equal_suffix*
by *(metis asm4)*
next
case *(STMap x1 tp)*
have **:" i < x ∧ (hash destl (ShowL$_{nat}$ x) = hash destl (ShowL$_{nat}$ i) ∨ (∃ ia. typeCon x1 ia ∧ TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash (hash destl (ShowL$_{nat}$ i)) ia) tp))"*
using *asm4 STMap* **unfolding** *TypedStoSubpref.simps* **by** *simp*
then have *"hash destl (ShowL$_{nat}$ x) ≠ hash destl (ShowL$_{nat}$ i)"* **using** *hash_injective ShowLNatDot Read_Show_nat'_id readLintNotEqual ** **by** *metis*
then obtain *ia* **where** *"(typeCon x1 ia ∧ TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash (hash destl (ShowL$_{nat}$ i)) ia) tp)"* **using** *** **by** *auto*
then have *a3:"TypedStoSubpref (hash destl (ShowL$_{nat}$ x)) (hash destl (hash (ShowL$_{nat}$ i) ia)) tp"*
by *(simp add: hash_suffixes_associative)*
then have *" (hash (ShowL$_{nat}$ i) ia) ≠ (ShowL$_{nat}$ x)"* **using** *hash_def ShowLNatDot*
by *(metis subPrefCannotBeInt)*
then show *?thesis* **using** *a3 hash_injective ShowLNatDot*
by *(metis TypedStoSubpref_child_imps_prnt TypedStoSubpref_hashes hash_never_equal_suffix)*
next
case *(STValue x')*
then have ***:"i < x ∧ hash destl (ShowL$_{nat}$ x) = hash destl (ShowL$_{nat}$ i)"* **unfolding** *TypedStoSubpref.simps* **using** *asm4* **by** *simp*
then have *"(ShowL$_{nat}$ x) = (ShowL$_{nat}$ i)"* **using** *hash_injective ShowLNatDot* **by** *blast*
then show *?thesis* **using** ****
by *(metis Read_Show_nat'_id readLintNotEqual)*
qed
qed

lemma *NoSubChanged_SCon_Preserved:*
assumes *"SCon t' destl srcMem"*
and *"∀ i' t''. accessStorage t'' (hash destl i') srcMem = accessStorage t'' (hash destl i') v'"*
and *"∀ t''. accessStorage t'' destl srcMem = accessStorage t'' destl v'"*
shows *"SCon t' destl v'"* **using** *assms*

```
proof(induction t' arbitrary: destl)
  case (STArray x1 t')
  then show ?case
    by (simp add: hash_suffixes_associative)
next
  case (STMap x1 t')
  then show ?case
    by (simp add: hash_suffixes_associative)
next
  case (STValue x)
  then show ?case by simp
qed


lemma SCon_NoChange:
  assumes "∀destl'. TypedStoSubpref destl' destl t' ⟶ (∀t. accessStorage t destl' srcMem =
accessStorage t destl' v')"
    and "SCon t' destl srcMem"
  shows "SCon t' destl v'" using assms
proof(induction t' arbitrary:destl)
  case (STArray x1 t')
  then show ?case
    by (meson SCon.simps(2) TypedStoSubpref.simps(2))
next
  case (STMap x1 t')
  then show ?case
    by (meson SCon.simps(3) TypedStoSubpref.simps(3))
next
  case (STValue x)
  then show ?case by simp
qed

lemma TypedStoSubpref_arr_longer:
  assumes "(∀subL. TypedStoSubpref subL srcl (STArray (Suc x) t') ⟶ ¬ TypedStoSubpref subL destl
(STArray (Suc x) t'))"
  shows "(∀subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL destl (STArray x
t'))"
proof intros
  fix subL
  assume "TypedStoSubpref subL srcl (STArray x t')"
  then have "TypedStoSubpref subL srcl (STArray (Suc x) t')" by auto
  then have "¬ TypedStoSubpref subL destl (STArray (Suc x) t')" using assms by blast
  then show " ¬ TypedStoSubpref subL destl (STArray x t')" by simp
qed

lemma SCon_preserved_disjoint_change:
  assumes "∀destl'. ¬ TypedStoSubpref destl' destl t' ⟶ (∀t. accessStorage t destl' srcMem =
accessStorage t destl' v')"
    and " ∀subL. TypedStoSubpref subL srcl t' ⟶ ¬ TypedStoSubpref subL destl t'"
    and " SCon t' srcl srcMem"
  shows "SCon t' srcl v'" using assms
proof(induction t' arbitrary:srcl)
  case (STArray x1 t')
  then show ?case
    by (meson SCon_NoChange)
next
  case (STMap x1 t')
  then show ?case
    by (meson SCon_NoChange)
next
  case (STValue x)
  then show ?case by simp
qed
```

```
lemma CompStoType_sharedSub:
  assumes "CompStoType pParentT tp1 pParentPtr p"
    and "TypedStoSubpref p stl2 tp2"
    and "CompStoType pParentT tp2 pParentPtr stl2"
  shows "CompStoType tp2 tp1 stl2 p " using assms
proof(induction tp2 arbitrary: stl2)
  case (STArray x1 tp2)
  then have cc:"p = stl2 ∨ (∃ i<x1. TypedStoSubpref p (hash stl2 (ShowL_nat i)) tp2)" unfolding
TypedStoSubpref.simps by simp
  then show ?case
  proof(cases "p = stl2")
    case True
    then show ?thesis
      using CompStoType_sameLocNdTyp CompStoType_sameLocs_sameType STArray.prems(2) STArray by blast
  next
    case False
    then show ?thesis
      by (meson CompStoType_sameLocNdTyp CompStoType_trns STArray.IH STArray.prems cc
CompStoType.simps(2))
  qed
next
  case (STMap x1 tp2)
  then have cc:"p = stl2 ∨ (∃ i. typeCon x1 i ∧ TypedStoSubpref p (hash stl2 i) tp2)"
    unfolding TypedStoSubpref.simps by simp
  then show ?case
  proof(cases "p = stl2")
    case True
    then show ?thesis
      using CompStoType_sameLocs_sameType STMap.prems by simp
  next
    case False
    then obtain i where idef:"(typeCon x1 i ∧ TypedStoSubpref p (hash stl2 i) tp2)" using cc by
blast
    then show ?thesis using STMap.IH[of "(hash stl2 i)"]
      by (meson CompStoType_sameLocNdTyp CompStoType_trns STMap.prems CompStoType.simps(3))
  qed
next
  case (STValue x)
  then show ?case
    by (simp add: CompStoType_sameLocs_sameType)
qed

lemma SCon_sub_imps_Parent:
  assumes " CompStoType struct (STArray x t''') x4 p'"
    and "SCon (STArray x t''') p' sto'"
    and "SCon struct x4 sto"
    and "t' = STArray x t''' "
    and "∀ locs t''. locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
       accessStorage t'' locs sto = accessStorage t'' locs sto'"
  shows "SCon struct x4 sto'" using assms(1,2,3)
proof(induction struct arbitrary: x4)
  case (STArray x1 struct)
  have c:"STArray x t''' = STArray x1 struct ∧ x4 = p' ∨ (∃ i<x1. CompStoType struct (STArray x t''')
(hash x4 (ShowL_nat i)) p')"
    using STArray(2) unfolding CompStoType.simps by blast

  then show ?case
  proof(cases "x4 = p'")
    case True
    then have "STArray x t''' = STArray x1 struct" using c
      using CompStoType_sameLoc_sameType STArray.prems(1) by blast
    then show ?thesis using assms True by auto
  next
```

```
    case False
    then obtain i where idef:"(i<x1 ∧ CompStoType struct (STArray x t''') (hash x4 (ShowL_nat i)) p')"
using c by auto
    then have sc:"SCon struct (hash x4 (ShowL_nat i)) sto'"
      using STArray.IH[of "(hash x4 (ShowL_nat i))"] STArray(3)
      using STArray.prems(3) by auto
    have " ∀i<x1. SCon struct (hash x4 (ShowL_nat i)) sto'"
    proof intros
      fix i'' assume *:"i''<x1"
      then show "SCon struct (hash x4 (ShowL_nat i'')) sto'"
      proof(cases "i'' = i")
        case True
        then show ?thesis using idef sc by simp
      next
        case False
        then have "¬CompStoType struct (STArray x t''') (hash x4 (ShowL_nat i'')) p'"
          using idef
          by (metis CompStoType_unique_location ShowLNatDot hashesIntSame)
        then have g:"¬TypedStoSubpref p' (hash x4 (ShowL_nat i'')) struct"
          by (metis CompStoType_imps_TypedStoSubpref NotRelatedPrnt_imps_notRelatedChild ShowLNatDot
              hash_adds_depth idef same_depth_imp_neg_TypedStoSubpref)
        have g5:"¬ TypedStoSubpref (hash x4 (ShowL_nat i'')) p' (STArray x t''')"
          by (metis CompStoType_imps_TypedStoSubpref STArray.prems(1) ShowLNatDot
              TypedStoSubpref_child_imps_prnt TypedStoSubpref_hashes g idef
              typedStoSub_imps_negInv)
        have g6:"∀l. ¬ TypedStoSubpref (hash (hash x4 (ShowL_nat i'')) l) p' (STArray x t''')"
        proof (intro allI notI)
          fix l
          assume "TypedStoSubpref (hash (hash x4 (ShowL_nat i'')) l) p' (STArray x t''')"
          then have "LSubPrefL2 (hash (hash x4 (ShowL_nat i'')) l) p'"
            using TypedStoSubpref_b unfolding LSubPrefL2_def by metis
          moreover have "LSubPrefL2 p' (hash x4 (ShowL_nat i))"
            using CompStoType_imps_subloc idef unfolding LSubPrefL2_def by metis
          ultimately have "LSubPrefL2 (hash (hash x4 (ShowL_nat i'')) l) (hash x4 (ShowL_nat i))"
            by (rule LSubPrefL2_trans)
          then show False using LSubPrefL2_index_unique LSubPrefL2_hash False by blast
        qed
        have g2:"SCon struct (hash x4 (ShowL_nat i'')) sto" using STArray(4)
          by (simp add: "*")
        then show ?thesis using assms CompStoType_imp_subType sublocs_nonchanged_SCon[OF g ] g5 g6 by
blast
      qed
    qed
    then show ?thesis unfolding SCon.simps by simp
  qed

next
  case (STMap x1 struct)
  obtain i where idef:"(typeCon x1 i ∧ CompStoType struct (STArray x t''') (hash x4 i) p')"
    using STMap(2) unfolding CompStoType.simps by blast
  then have sc:"SCon struct (hash x4 i) sto'"
    using STMap.IH[of "(hash x4 i)"] STMap(3)
    using STMap.prems(3) by auto
  have "∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) sto'"
  proof intros
    fix i'' assume *:"typeCon x1 i''"
    then show "SCon struct (hash x4 i'') sto'"
    proof(cases "i'' = i")
      case True
      then show ?thesis using idef sc by simp
    next
      case False
      then have "¬CompStoType struct (STArray x t''') (hash x4 i'') p'" using idef
        using "*" CompStoType_unique_location typeConNoDots by blast
```

```
      then have g:"¬TypedStoSubpref p' (hash x4 i'') struct"
        by (metis <¬ CompStoType struct (STArray x t''') (hash x4 i'') p'> assms(4) STMap.prems(1) "*"
CompStoType_sharedSub CompStoType.simps(3) CompStoType_sameLocNdTyp)
      have g5:"¬ TypedStoSubpref (hash x4 i'') p' (STArray x t''')"
        by (metis g assms(4) TypedStoSubpref_sameLoc STMap.prems(1) typedStoSub_imps_negInv
typeConNoDots "*" CompStoType_imps_subloc idef
            TypedStoSubpref_child_imps_prnt CompStoType_imps_TypedStoSubpref hash_flatten_right
hash_inequality)
      have g6:"∀ l. ¬ TypedStoSubpref (hash (hash x4 i'') l) p' (STArray x t''')"
      proof (intro allI notI)
        fix l
        assume "TypedStoSubpref (hash (hash x4 i'') l) p' (STArray x t''')"
        then have "LSubPrefL2 (hash (hash x4 i'') l) p'"
          using TypedStoSubpref_b unfolding LSubPrefL2_def by metis
        moreover have "LSubPrefL2 p' (hash x4 i)"
          using CompStoType_imps_subloc idef unfolding LSubPrefL2_def by metis
        ultimately have "LSubPrefL2 (hash (hash x4 i'') l) (hash x4 i)"
          by (rule LSubPrefL2_trans)
        then show False using LSubPrefL2_hash_unique idef "*" typeConNoDots LSubPrefL2_hash False by
blast
      qed
      have g2:"SCon struct (hash x4 i'') sto" using STMap(4)
        by (simp add: "*")
      then show ?thesis using assms CompStoType_imp_subType sublocs_nonchanged_SCon[OF g assms(5) g2]
g5 g6 by blast
    qed
  qed
  then show ?case unfolding SCon.simps
    by simp
next
  case (STValue x)
  then show ?case by simp
qed

lemma SCon_update_array_subloc_cases:
  assumes cmp:
      "if TypedStoSubpref p' x4 struct then CompStoType struct t' x4 p'
       else if TypedStoSubpref x4 p' t' then CompStoType t' struct p' x4 else True"
    and scParent: "SCon (STArray x t''') p' sto'"
    and scChanged: "SCon t' p' sto'"
    and scOld: "SCon struct x4 sto"
    and tDef: "t' = STArray x t'''"
    and unchanged:
      "∀ locs t''. locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
         accessStorage t'' locs sto = accessStorage t'' locs sto'"
  shows "SCon struct x4 sto'"
proof (cases "TypedStoSubpref p' x4 struct")
  case True
  then have "CompStoType struct (STArray x t''') x4 p'"
    using cmp tDef by simp
  then show ?thesis
    using SCon_sub_imps_Parent[of struct x t''' x4 p' sto' sto t'] scParent scOld tDef unchanged by
blast
next
  case False
  then show ?thesis
  proof (cases "TypedStoSubpref x4 p' t'")
    case True
    then have "CompStoType t' struct p' x4" using cmp False by simp
    then show ?thesis using scChanged SCon_imps_sublocs by blast
  next
    case False2: False
    show ?thesis using sublocs_nonchanged_SCon[OF False unchanged scOld] False2 tDef by simp
  qed
```

qed

Updating the storage array at `l` preserves *SCon* at a location `x4` that shares a parent `lr` with `l`.

```
lemma SCon_preserved_array_update_related:
  assumes rel: "CompStoType tr (STArray x t) lr l"
    and cmp: "CompStoType tr struct lr x4"
    and new: "SCon (STArray x t) l sto'"
    and old: "SCon struct x4 sto"
    and same: "∀ locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t) ⟶
      accessStorage t'' locs sto = accessStorage t'' locs sto'"
  shows "SCon struct x4 sto'"
proof (cases "TypedStoSubpref x4 l (STArray x t)")
  case True
  have "CompStoType (STArray x t) struct l x4"
    using CompStoType_sharedSub[OF cmp True rel] .
  then show ?thesis using SCon_imps_sublocs new by blast
next
  case nx: False
  show ?thesis
  proof (cases "TypedStoSubpref l x4 struct")
    case True
    have "CompStoType struct (STArray x t) x4 l"
      using CompStoType_sharedSub[OF rel True cmp] .
    then show ?thesis using SCon_sub_imps_Parent[OF _ new old refl same] by blast
  next
    case False
    show ?thesis using sublocs_nonchanged_SCon[OF False same old nx] .
  qed
qed
```

The same holds when `x4` and the parent `lr` of `l` are unrelated.

```
lemma SCon_preserved_array_update_unrelated:
  assumes rel: "CompStoType tr (STArray x t) lr l"
    and n1: "¬ TypedStoSubpref x4 lr tr"
    and n2: "¬ TypedStoSubpref lr x4 struct"
    and old: "SCon struct x4 sto"
    and same: "∀ locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t) ⟶
      accessStorage t'' locs sto = accessStorage t'' locs sto'"
  shows "SCon struct x4 sto'"
  using sublocs_nonchanged_SCon[OF
      NotRelatedPrnt_imps_notRelatedChild[OF n1 n2 CompStoType_imps_TypedStoSubpref[OF rel]]
      same old NotReachablePrnt_imps_notReachableChild[OF n1 n2 rel]] .
```

Combined form, taking the pointer-compatibility fact from *compPointers*.

```
lemma SCon_preserved_array_update:
  assumes rel: "CompStoType tr (STArray x t) lr l"
    and ptr: "if TypedStoSubpref lr x4 struct then CompStoType struct tr x4 lr
      else if TypedStoSubpref x4 lr tr then CompStoType tr struct lr x4 else True"
    and n2: "¬ TypedStoSubpref lr x4 struct"
    and new: "SCon (STArray x t) l sto'"
    and old: "SCon struct x4 sto"
    and same: "∀ locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t) ⟶
      accessStorage t'' locs sto = accessStorage t'' locs sto'"
  shows "SCon struct x4 sto'"
proof (cases "TypedStoSubpref x4 lr tr")
  case True
  then have "CompStoType tr struct lr x4" using ptr n2 by simp
  then show ?thesis using SCon_preserved_array_update_related[OF rel _ new old same] by blast
next
  case False
  show ?thesis using SCon_preserved_array_update_unrelated[OF rel False n2 old same] .
qed
```

The pointer-compatibility condition carries over from a location `p` to a sub-location `l1` of `p`.

```
lemma compPointers_if_child:
  assumes comps: "if TypedStoSubpref p l2 t2 then CompStoType t2 tp l2 p
                  else if TypedStoSubpref l2 p tp then CompStoType tp t2 p l2 else True"
    and child: "CompStoType tp t1 p l1"
  shows "if TypedStoSubpref l2 l1 t1 then CompStoType t1 t2 l1 l2
         else if TypedStoSubpref l1 l2 t2 then CompStoType t2 t1 l2 l1 else True"
proof (cases "l1 = l2")
  case True
  then show ?thesis using comps child
    by (metis CompStoType_imps_TypedStoSubpref CompStoType_sameLocNdTyp CompStoType_sharedSub)
next
  case ne: False
  show ?thesis
  proof (cases "TypedStoSubpref l2 l1 t1")
    case True
    then show ?thesis using comps child ne
      by (metis CompStoType_imps_TypedStoSubpref CompStoType_sharedSub CompStoType_trns
          NotReachablePrnt_imps_notReachableChild typedStoSub_imps_negInv)
  next
    case nsub: False
    show ?thesis
    proof (cases "TypedStoSubpref l1 l2 t2")
      case True
      then show ?thesis using comps child ne nsub
        by (metis CompStoType_imps_TypedStoSubpref CompStoType_sharedSub CompStoType_trns
            NotRelatedPrnt_imps_notRelatedChild)
    next
      case False
      then show ?thesis using nsub by auto
    qed
  qed
qed

end
```

7 Type consistency for environments and states (TypeSafe_Def)

```
theory TypeSafe_Def
  imports TypeSafe_Memory TypeSafe_Storage

begin
```

7.1 Ensuring unique locations

No two entries of a `Denvalue` map share a location.

```
definition unique_locations :: "(String.literal, type × denvalue) fmap ⇒ bool" where
  "unique_locations denval = (∀x y. x |∈| fmran denval ∧ y |∈| fmran denval ∧ snd x = snd y ⟶ x =
y)"
```

If an environment has unique locations, two entries of its `Denvalue` with the same location are equal. Hence no two variables share a location.

```
lemma uniqueLocs:
  assumes "unique_locations (Denvalue ev)"
    and "x |∈| fmran (Denvalue ev)"
    and "y |∈| fmran (Denvalue ev)"
    and "snd x = snd y"
  shows "x = y" using assms unique_locations_def by blast
```

7.1.1 Pushing to the stack maintains stackLocations

```
lemma stackPushToplocSafe:
  assumes "lessThanTopLocs k"
    and "k' = push p k"
```

```
  shows "lessThanTopLocs k'" unfolding lessThanTopLocs_def
proof(intros)
  fix tloc loc
  assume **:"Toploc k' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
  have a120:"Toploc k<Toploc k'" using assms by (simp add:push_def allocate_def updateStore_def
accessStore_def)
  have a125:"tloc ≠ Toploc k" using "**" a120 by auto
  then have a126:"(ShowL_nat tloc) ≠ (ShowL_nat (Toploc k))"
    by (metis Read_Show_nat'_id)
  have a130:"∀tloc loc. Toploc k ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc k =
None"
    using assms(1) unfolding lessThanTopLocs_def by simp
  have a140:"k' = (let s = updateStore (ShowL_nat (Toploc k)) p k in snd (allocate s))" unfolding
push_def assms(2) by auto
  then have acces:"∀locs. locs ≠ (ShowL_nat (Toploc k)) ⟶ accessStore locs k' = accessStore locs k"
unfolding accessStore_def updateStore_def
    by (simp add:push_def allocate_def updateStore_def accessStore_def split:if_split_asm)
  have a150:"accessStore (ShowL_nat tloc) k = None" using a130 a120 **
    by (metis LSubPrefL2_def dual_order.strict_trans1 nless_le)
  then show "accessStore loc k' = None"
  proof(cases "loc = (ShowL_nat tloc)")
    case True
    then show ?thesis using acces a150 a126 by simp
  next
    case False
    then show ?thesis using acces a150 a126 ShowLNatDot
      by (metis "**" LSubPrefL2_def a120 a130 hash_inequality hash_int_prefix hash_suffixes_associative
nat_less_le order_less_le_trans)
  qed
next
  fix loc y
  assume *:"accessStore loc k' = Some y"
  have a140:"k' = (let s = updateStore (ShowL_nat (Toploc k)) p k in snd (allocate s))" unfolding
push_def assms(2) by auto
  then have acces:"∀locs. locs ≠ (ShowL_nat (Toploc k)) ⟶ accessStore locs k' = accessStore locs k"
unfolding accessStore_def updateStore_def
    by (simp add:push_def allocate_def updateStore_def accessStore_def split:if_split_asm)
  have " ∃tloc<Toploc k'. LSubPrefL2 loc (ShowL_nat tloc)"
  proof(cases "loc = (ShowL_nat (Toploc k))")
    case True
    have a10:"Toploc k < Toploc k'" using a140 unfolding allocate_def updateStore_def by simp
    have "LSubPrefL2 loc (ShowL_nat (Toploc k))" using True LSubPrefL2_def by simp
    have "∀l. l≠(ShowL_nat (Toploc k)) ⟶ ¬(LSubPrefL2
        (ShowL_nat (Toploc k)) l)" using ShowLNatDot unfolding LSubPrefL2_def hash_def
      using subPrefCannotBeInt by auto
    then show ?thesis using True a10 assms
      using LSubPrefL2_def by auto
  next
    case False
    have "((∀tloc loc. Toploc k ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc k = None)
          ∧ (∀loc y. accessStore loc k = Some y ⟶ (∃tloc<Toploc k. LSubPrefL2 loc (ShowL_nat
tloc))))"
      using assms lessThanTopLocs_def[of k] by blast
    then obtain tloc where tlocdef:"tloc < Toploc k ∧ LSubPrefL2 loc (ShowL_nat tloc)" using False *
acces by metis
    moreover have "Toploc k < Toploc k'" using a140 unfolding allocate_def updateStore_def by simp
    ultimately show ?thesis using assms lessThanTopLocs_def[of k] * tlocdef
      using order.strict_trans by auto
  qed
  then show "∃tloc<Toploc k'. LSubPrefL2 loc (ShowL_nat tloc)" by simp
qed

lemma lessThanSome_imps_Locs:
  assumes "lessThanTopLocs mem"
```

```
    and "accessStore p' mem = Some y"
    and "LSubPrefL2 p' (ShowL_nat x)"
  shows "x < Toploc mem"
proof(cases "p' = (ShowL_nat x)")
  case True
  have a10:"∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None"
    using assms(1) unfolding lessThanTopLocs_def by simp

  have a20:"accessStore (ShowL_nat x) mem = Some y" using assms(2) True by simp
  have "ReadL_nat(ShowL_nat x) = x" using Read_Show_nat'_id by auto
  have "x < Toploc mem" using a10 a20
    by (metis LSubPrefL2_def linorder_le_less_linear option.discI)
  then have "LSubPrefL2 (ShowL_nat x) (ShowL_nat x) ∧ x < Toploc mem" unfolding LSubPrefL2_def by simp
  then show ?thesis using True by simp
next
  case False
  have a10:"∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None"
    using assms(1) unfolding lessThanTopLocs_def by simp
  then obtain i where a20:"(p' = hash (ShowL_nat x) i)" using assms(3) False unfolding LSubPrefL2_def
by blast
  then have a20:"accessStore (hash (ShowL_nat x) i) mem = Some y" using assms(2) by simp
  have "x < Toploc mem" using a10 a20
    by (metis LSubPrefL2_def linorder_le_less_linear option.discI)
  then show ?thesis by blast
qed

lemma lessThanSome_imps_Locs_hash:
  assumes "lessThanTopLocs mem"
    and " accessStore (hash p' (ShowL_nat i)) mem = Some y"
    and "LSubPrefL2 p' (ShowL_nat x)"
  shows "x < Toploc mem"
proof(cases "p' = (ShowL_nat x)")
  case True
  have a10:"∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None"
    using assms(1) unfolding lessThanTopLocs_def by simp

  have a20:"accessStore (hash p' (ShowL_nat i)) mem = Some y" using assms(2) True by simp
  have "ReadL_nat(ShowL_nat x) = x" using Read_Show_nat'_id by auto
  have "x < Toploc mem" using a10 a20
    by (metis LSubPrefL2_def True assms(1) lessThanSome_imps_Locs)
  then have "LSubPrefL2 (ShowL_nat x) (ShowL_nat x) ∧ x < Toploc mem" unfolding LSubPrefL2_def by simp
  then show ?thesis using True by simp
next
  case False
  have a10:"∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None"
    using assms(1) unfolding lessThanTopLocs_def by simp
  then obtain i' where a20:"(p' = hash (ShowL_nat x) i')" using assms(3) False unfolding
LSubPrefL2_def by blast
  then have a20:"accessStore (hash (hash (ShowL_nat x) i') (ShowL_nat i)) mem = Some y" using assms(2)
by simp
  have "x < Toploc mem" using a10 a20
    by (metis LSubPrefL2_def assms(1) hash_suffixes_associative lessThanSome_imps_Locs)
  then show ?thesis by blast
qed

lemma subPtrs_nonTop:
  assumes "MCon (MTArray x11' x12') mem stl1"
    and "(∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None)"
    and "(∀loc y. accessStore loc mem = Some y ⟶ (∃tloc<Toploc mem. LSubPrefL2 loc (ShowL_nat
```

```
tloc)))"
    and "∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem)) ⟶ accessStore locs mem' = accessStore locs
mem"
    and "TypedMemSubPrefPtrs mem' x11' x12' stl1 dloc1"
  shows "¬ LSubPrefL2 dloc1 (ShowL_nat (Toploc mem))"
proof(cases x12')
  case (MTArray x11 x12)
  then obtain i l where idef:"i<x11' ∧ accessStore (hash stl1 (ShowL_nat i)) mem' = Some (MPointer l)
∧ (l = dloc1 ∨ TypedMemSubPrefPtrs mem' x11 x12 l dloc1)"
    using assms by auto
  have "¬ LSubPrefL2 (hash stl1 (ShowL_nat i)) (ShowL_nat(Toploc mem))" using assms(1,2) LSubPrefL2_def
    by (metis MConArrayPointers MTArray gr_zeroI idef le_refl less_nat_zero_code option.discI)
  then have mconl:"MCon (MTArray x11 x12) mem l" using assms(1,4) MTArray idef
    by (metis MCon_imps_sub_Mcon)
  then have lsubloc:"¬ LSubPrefL2 l (ShowL_nat (Toploc mem))" using assms(2) LSubPrefL2_def
hash_suffixes_associative mconl idef
    by (metis (no_types, lifting) MCon_imps_Some le_refl option.discI)
  show ?thesis
  proof
    assume *:"LSubPrefL2 dloc1 (ShowL_nat (Toploc mem))"
    show False
    proof(cases "l = dloc1")
      case True
      then show ?thesis using "*" lsubloc by blast
    next
      case False
      then have lneq2:"∀ i<x11. hash l (ShowL_nat i) ≠ dloc1" using mconl lsubloc idef assms(3,4)
        using "*" MemLSubPrefL2_specific_imps_general by blast
      then have "TypedMemSubPrefPtrs mem' x11 x12 l dloc1" using idef False by blast
      then show ?thesis using lneq2 lsubloc mconl
      proof(induction x12 arbitrary:x11 l)
        case (MTArray x1 x12')
        have "∃ i'<x11. ∃ l'. accessStore (hash l (ShowL_nat i')) mem' = Some (MPointer l') ∧ (l' = dloc1
∨ TypedMemSubPrefPtrs mem' x1 x12' l' dloc1)"
          using MTArray(2) unfolding TypedMemSubPrefPtrs.simps by simp
        then obtain i' l' where i'def:"i'<x11 ∧ accessStore (hash l (ShowL_nat i')) mem' = Some
(MPointer l')
          ∧ (l' = dloc1 ∨ TypedMemSubPrefPtrs mem' x1 x12' l' dloc1)" by blast
        then have mconl':"MCon (MTArray x1 x12') mem l'" using assms(4) MTArray(5)
          by (metis MCon_imps_sub_Mcon MTArray.prems(3) MemLSubPrefL2_specific_imps_general)
        have l'neq:"l' ≠ dloc1"
          using "*" AllPtrsNotTop_MCon assms(2,3) lessThanTopLocs_def mconl' by blast
        then have "TypedMemSubPrefPtrs mem' x1 x12' l' dloc1" using i'def by simp
        then have l'subloc:"¬ LSubPrefL2 l' (ShowL_nat (Toploc mem))" using l'neq assms i'def mconl'
          using AllPtrsNotTop_MCon lessThanTopLocs_def by blast
        moreover have "∀ i<x1. hash l' (ShowL_nat i) ≠ dloc1" using l'subloc
          using "*" MemLSubPrefL2_specific_imps_general by blast
        ultimately show ?case using MTArray.IH[of x1 l'] i'def
          using l'neq mconl' by blast
      next
        case (MTValue x)
        then show ?case by auto
      qed
    qed
  qed
next
  case (MTValue x2)
  then show ?thesis using assms(1,2,5) by fastforce
qed

lemma PreExistMconNotChangeByToploc:
  assumes "(∀ loc y. accessStore loc mem = Some y ⟶ (∃ tloc<Toploc mem. LSubPrefL2 loc (ShowL_nat
tloc)))"
    and "(∀ tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
```

```
None)"
    and "∀ locs. ¬ LSubPrefL2 locs (ShowL_nat(Toploc mem)) ∨ locs =
        (ShowL_nat(Toploc mem)) ⟶ accessStore locs mem = accessStore locs m"
    and "MCon tp1 mem stl1"
  shows "MCon tp1 m stl1" using assms(4)
proof(induction tp1 arbitrary: stl1)
  case (MTArray x1 struct)
  then have accessSome:"∀ i'<x1. ∃ y. accessStore (hash stl1 (ShowL_nat i')) mem = Some y"
    by (metis bot_nat_0.not_eq_extremum less_nat_zero_code mcon_accessStore)
  moreover have accessSame:"∀ i'<x1. accessStore (hash stl1 (ShowL_nat i')) mem = accessStore (hash
stl1 (ShowL_nat i')) m" using calculation
    by (metis Read_Show_nat'_id antisym_conv1 assms(2,3) option.distinct(1) readLintNotEqual)

  have "∀ i<x1.
(case accessStore (hash stl1 (ShowL_nat i)) m of None ⇒ False
  | Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct m (hash stl1
(ShowL_nat i)))
  | Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct m loc2 | MTValue Types ⇒
False))"
  proof intros
    fix i' assume asm3:"i'<x1"
    show "case accessStore (hash stl1 (ShowL_nat i')) m of None ⇒ False
        | Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct m
(hash stl1 (ShowL_nat i')))
        | Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct m loc2 | MTValue
Types ⇒ False)"
    proof(cases "accessStore (hash stl1 (ShowL_nat i')) m")
      case None
      then show ?thesis using accessSome accessSame asm3 by auto
    next
      case (Some a)
      then show ?thesis
      proof(cases a)
        case (MValue x1')
        then show ?thesis using accessSome accessSame Some asm3 MTArray by (cases struct; auto)
      next
        case (MPointer x2)
        then have "MCon struct mem x2" using accessSome accessSame Some asm3 MTArray
          by (metis MconSameTypeSameAccessWithTyping)
        then have "MCon struct m x2" using MTArray by simp
        then show ?thesis using accessSome accessSame Some asm3 MTArray MPointer by (cases struct;
auto)
      qed
    qed
  qed
  moreover have "x1>0" using MTArray(2) unfolding MCon.simps by presburger
  moreover have "(∃ p. accessStore stl1 m = Some (MPointer p)) ∨ accessStore stl1 m = None" using
calculation(3) MTArray unfolding MCon.simps
    by (metis accessSome lessThanSome_imps_Locs_hash lessThanTopLocs_def assms(1,2,3) nat_less_le )
  ultimately show ?case using MCon.simps(2)[of x1 struct m ] by simp
next
  case (MTValue x')
  then show ?case
    by (metis MCon.simps(1) assms(2,3) case_optionE le_refl option.discI)
qed
```

7.2 Balances and message values conform

```
definition balanceTypes :: "accounts ⇒ bool" where
  "balanceTypes acc = (∀ adv. typeCon (TUInt b256) (Bal (acc adv)))"

definition addressFormat::"address ⇒ bool"
  where "addressFormat cur = (typeCon TAddr cur)"
```

abbreviation *envAddressesWellFormed :: "environment ⇒ bool"* **where**
"envAddressesWellFormed ev ≡ addressFormat (Address ev) ∧ addressFormat (Sender ev)"

definition *svalueTypes :: "valuetype ⇒ bool"* **where**
"svalueTypes sval = typeCon (TUInt b256) sval"

7.3 Compatible pointers

Arguments: parent type, parent location, sub-prefix type, sub-prefix location.

primrec *Subpref :: "stypes ⇒ location ⇒ stypes ⇒ location ⇒ bool"*
where
"Subpref (STValue typ) Modulatedparent typ2 subloc = ((Modulatedparent = subloc) ∧ typ2 = (STValue typ))"
|"Subpref (STArray len arr) Modulatedparent typ2 subloc = (((STArray len arr) = typ2 ∧ Modulatedparent = subloc)
∨ (∃i::nat. i < len ∧ Subpref arr (hash Modulatedparent (ShowL$_{nat}$ i)) typ2 subloc))"
|"Subpref (STMap fromTyp toTyp) Modulatedparent typ2 subloc = (((STMap fromTyp toTyp) = typ2 ∧ Modulatedparent = subloc)
∨(∃i::String.literal. (typeCon fromTyp i) ∧ Subpref toTyp (hash Modulatedparent i) typ2 subloc))"

lemma *"¬Subpref (STArray 3 (STValue TBool)) (STR ''1.5'') (STValue TBool) (STR ''4.1.5'')"* **using** *Subpref.simps hash_def* **by** *simp? eval*
lemma *"Subpref (STArray 5 (STValue TBool)) (STR ''1.5'') (STValue TBool) (STR ''1.1.5'')"* **using** *Subpref.simps hash_def* **by** *simp? eval*
lemma *"Subpref (STArray 5 (STArray 4 (STArray 3 (STValue TBool)))) (STR ''1.5'') ((STValue TBool)) (STR ''1.1.1.1.5'') "* **using** *Subpref.simps hash_def* **by** *simp? eval*
lemma *"Subpref (STArray 5 (STArray 4 (STArray 3 (STValue TBool)))) (STR ''1.5'') ((STArray 3 (STValue TBool))) (STR ''1.1.1.5'') "* **using** *Subpref.simps hash_def* **by** *simp? eval*

The value stored for one variable of the environment conforms to its declared type.

abbreviation *denvalueCon :: "stack ⇒ memoryT ⇒ storageT ⇒ calldataT ⇒ type ⇒ denvalue ⇒ bool"*
where *"denvalueCon sck mem sto cd t l ≡*
(case l of Stackloc loc ⇒
(case accessStore loc sck of None ⇒ False
| Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
| Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
| Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct mem stloc | _ ⇒ False)
| Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc sto | _ ⇒ False))
| Storeloc loc ⇒ (case t of type.Storage typ ⇒ SCon typ loc sto | _ ⇒ False))"

definition *typeCompat::"(String.literal, type × denvalue) fmap ⇒ stack ⇒ memoryT ⇒ storageT ⇒ calldataT ⇒ bool"*
where *"typeCompat den sck0 mem sto cd = (∀t l. (t, l) |∈| fmran (den) ⟶ denvalueCon sck0 mem sto cd t l)"*

7.4 Supplementary lemmas

Collection of lemmas which shorten common proof obligations

Support functions to simplify the typeCon expressions definition

fun *extractValueType :: "stackvalue ⇒ valuetype"* **where**
"extractValueType (KValue v) = v"
| "extractValueType (KCDptr v) = v"
| "extractValueType (KMemptr v) = v"
| "extractValueType (KStoptr v) = v"

A value of a given type is consistent with calldata, memory, and storage.

abbreviation *valueCon :: "environment ⇒ calldataT ⇒ state ⇒ stackvalue ⇒ type ⇒ bool"*
where *"valueCon ev cd st v t ≡*
(case t of Value typ ⇒ typeCon typ (extractValueType v)

```
    | Calldata struct ⇒ MCon struct cd (extractValueType v)
    | type.Memory struct ⇒ MCon struct (Memory st) (extractValueType v)
    | type.Storage struct ⇒ SCon struct (extractValueType v) (Storage st (Address ev)))"
```

The result of evaluating an expression is consistent with its type and, for pointers, is related to a variable of the environment: calldata and memory pointers reach the value through a stack variable, storage pointers through a storage variable.

```
abbreviation exprResultCon :: "environment ⇒ calldataT ⇒ state ⇒ stackvalue ⇒ type ⇒ bool"
  where "exprResultCon ev cd st v t ≡
    (case t of Value typ ⇒ typeCon typ (extractValueType v) ∧ (∃xx. v = KValue xx)
     | Calldata struct ⇒ MCon struct cd (extractValueType v) ∧ (∃xx. v = KCDptr xx) ∧
         (∃stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧
            accessStore stloc (Stack st) = Some (KCDptr p) ∧
            (tp'' = struct ∧ v = KCDptr p ∨ (∃len arr. extractValueType v ≠ p ∧
               tp'' = MTArray len arr ∧ CompMemType cd len arr struct p (extractValueType v))))
     | type.Memory struct ⇒ MCon struct (Memory st) (extractValueType v) ∧ (∃xx. v = KMemptr xx) ∧
         (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧
            accessStore stloc (Stack st) = Some (KMemptr p) ∧
            (tp'' = struct ∧ v = KMemptr p ∨ (∃len arr. extractValueType v ≠ p ∧
               tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr struct p (extractValueType
v))))
     | type.Storage struct ⇒ SCon struct (extractValueType v) (Storage st (Address ev)) ∧
         (∃xx. v = KStoptr xx) ∧
         (∃stloc tp''. (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue ev) ∧
            (tp'' = struct ∧ v = KStoptr stloc ∨
             extractValueType v ≠ stloc ∧ CompStoType tp'' struct stloc (extractValueType v))))"


locale typesafe_base = statement_with_gas +
  assumes methodVarsNoPref: "∀c ct dud i1 i2 t1 t2. i1 ≠ i2 ∧ ep $$ c = Some(ct,dud) ∧ ct $$ i1 =
Some (Var t1) ∧ ct $$ i2 = Some (Var t2)
                              ⟶ (¬TypedStoSubpref i1 i2 t2) ∧ (¬TypedStoSubpref i2 i1 t1)"

context typesafe_base
begin
```

An environment and state are fully initialised if the contract has an initialised account and all contract variables are bound in the *Denvalue*. These are the first steps when creating a new environment (see statement *NEW*).

```
definition fullyInitialised::"environment ⇒ accounts ⇒ stack ⇒ bool"
  where "fullyInitialised env acc sk = (
(∃c ct dud.
          Type (acc (Address env)) = Some (atype.Contract c) ∧
          Contract env = c ∧
          ep $$ c = Some (ct, dud) ∧
          (∀id v. ct $$ id = Some (Var v) ⟷
                  Denvalue env $$ id = Some (type.Storage v, Storeloc id)) ∧
          (∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧

          (∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue env) ∧
              accessStore l sk = Some (KStoptr p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧
                       CompStoType t' t l' p)))


)"

lemma test_not_vacuous:
  fixes my_contract :: "String.literal"
  fixes n :: "stypes"
  assumes blueprint_def: "my_blueprint = fmupd (STR ''balance'') (Var n) fmempty"
  assumes ep_def: "ep $$ my_contract = Some (my_blueprint, dud)"

  shows "∃env acc sk. fullyInitialised env acc sk ∧
```

```
                    Denvalue env $$ (STR ''balance'') = Some (type.Storage n, Storeloc (STR
''balance''))"
proof -
  let ?addr = "STR ''A''"
  let ?den = "fmupd (STR ''balance'') (type.Storage n, Storeloc (STR ''balance'')) fmempty"
  let ?env = "(emptyEnv ?addr my_contract (STR '''') (STR ''''))(|Denvalue := ?den|)"
  let ?acc = "emptyAccount(?addr := emptyAcc(|Type := Some (atype.Contract my_contract)|))"

  have map_eq: "⋀i v. my_blueprint $$ i = Some (Var v) ⟷
                   Denvalue ?env $$ i = Some (type.Storage v, Storeloc i)"
    using blueprint_def by simp

  have loc_eq: "⋀id v loc.
      Denvalue ?env $$ id = Some (type.Storage v, Storeloc loc) ⟹ id = loc"
    using fmlookup by (simp split:if_splits)

  have ptr_vacuous: "⋀t l p.
      (type.Storage t, Stackloc l) |∈| fmran (Denvalue ?env) ∧
      accessStore l emptyStore = Some (KStoptr p)
      ⟹ (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue ?env) ∧ CompStoType t' t l' p)"
    by (simp add: accessStore_def emptyStore_def)

  have fi: "fullyInitialised ?env ?acc emptyStore"
    unfolding fullyInitialised_def
    using ep_def map_eq loc_eq ptr_vacuous
    by (intro exI[of _ my_contract] exI[of _ my_blueprint] exI[of _ dud]) simp

  moreover have "Denvalue ?env $$ (STR ''balance'') =
      Some (type.Storage n, Storeloc (STR ''balance''))"
    by simp
  ultimately show ?thesis
    by blast
qed


definition AddressTypes :: "accounts ⇒ bool" where
    "AddressTypes ac =
      (∀adv. case Type (ac adv) of
               Some (atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep
             | _ ⇒ True)"

abbreviation accountsWellTyped :: "accounts ⇒ bool" where
  "accountsWellTyped acc ≡ balanceTypes acc ∧ AddressTypes acc"


definition safeContract :: "accounts ⇒ (address ⇒ storageT) ⇒ bool" where
  "safeContract acc sto =
    (∀e ct dud i tp.
      Type (acc (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
      ep $$ Contract e = Some (ct, dud) ∧
      fmlookup ct i = Some (Var tp)
      ⟶ SCon tp i (sto (Address (e::environment))))"
```

safeContract depends on the accounts only through their types.

```
lemma safeContract_Type_cong:
  assumes "safeContract acc sto"
    and "∀a. Type (acc' a) = Type (acc a)"
  shows "safeContract acc' sto"
  using assms unfolding safeContract_def by simp
```

7.5 Type consistency for reachable locations

This property ensures that the types stored in the typed mapping of memory are consistent with the types derivable from the denvalue for locations that are reachable from variables in the environment.

```
definition denvalueTypeCorrectness :: "environment ⇒ stack ⇒ memoryT ⇒ bool"
  where
    "denvalueTypeCorrectness env sck mem =
    (∀ t l ptr_loc. (type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
        accessStore l sck = Some (KMemptr ptr_loc) ⟶
        (case t of MTArray len arr ⇒
                (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem = Some arr)
          | MTValue val ⇒ accessTypeStore ptr_loc mem = Some (MTValue val))
      )"

definition subPrefixStructuralConsistency :: "memoryT ⇒ bool"
  where
    "subPrefixStructuralConsistency mem =
      (∀ locs tp. accessTypeStore locs mem = Some tp ⟶
          (case accessStore locs mem of Some (MPointer p) ⇒
               (∃ len arr. MCon tp mem p ∧ tp = MTArray len arr
                  ∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) mem = Some arr)
                )
           | Some (MValue v) ⇒
               (∃ val. MCon tp mem locs ∧ tp = MTValue val ∧accessTypeStore locs mem = Some tp)
           | None ⇒ False))"

definition SomeValSomeTyp:: "memoryT ⇒ bool"
  where
    "SomeValSomeTyp mem = (∀ locs. (∃ t. accessStore locs mem = Some t) ⟷ (∃ tt. accessTypeStore locs
mem = Some tt))"
```

7.6 Typesafe environments

TypeSafe states that an environment, accounts, stack, memory, storage, and calldata are mutually type-consistent. It combines two groups of properties:

1. structural properties of the environment and stores, such as unique locations, compatible pointers, and bounded locations;
2. value conformance: for every variable in the *Denvalue*, the string representation of its value conforms to its declared type.

```
definition TypeSafe :: "environment ⇒ accounts ⇒ stack ⇒ memoryT ⇒ (address ⇒ storageT) ⇒
calldataT ⇒ bool"
  where "TypeSafe ev acc sck mem sto cd= (
          typeCompat (Denvalue ev) sck mem (sto (Address ev)) cd ∧
          (unique_locations (Denvalue ev)) ∧
          (compPointers sck (Denvalue ev)) ∧
          (lessThanTopLocs sck) ∧ (lessThanTopLocs mem) ∧ (lessThanTopLocs cd) ∧
          envAddressesWellFormed ev ∧
          accountsWellTyped acc ∧
          (safeContract acc sto) ∧
          (svalueTypes (Svalue ev)) ∧
          denvalueTypeCorrectness ev sck mem ∧
          subPrefixStructuralConsistency mem ∧
          SomeValSomeTyp mem)"
```

Memory of a type-safe state holds values only below its top location.

```
lemma TypeSafe_memory_bounds:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "∀ loc y. accessStore loc mem = Some y ⟶ (∃ tloc<Toploc mem. LSubPrefL2 loc (ShowL_nat tloc))"
    and "∀ tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc mem =
None"
  using assms unfolding TypeSafe_def lessThanTopLocs_def by simp_all
```

7.7 All stack locations in the denvalue have a value in the stack

```
lemma typeSafeAllStacklocsExist:
```

```
  assumes "TypeSafe ev acc sck mem sto cd"
  assumes "(t, l2) |∈| fmran (Denvalue ev)"
    and "l2 = Stackloc l"
  shows "∃v. accessStore l sck = Some v"
proof -
  have "∀t l. (t, l) |∈| fmran (Denvalue ev) ⟶
        (case l of (Stackloc loc) ⇒
          (case (accessStore loc sck) of
            Some(KValue val) ⇒ (case t of (Value typ) ⇒ typeCon typ val | _ ⇒ False )
          | Some(KMemptr stloc) ⇒ (case t of (type.Memory struct) ⇒ MCon struct mem stloc | _ ⇒
False)
          | Some(KCDptr stloc) ⇒ (case t of (Calldata struct) ⇒ MCon struct cd stloc | _ ⇒ False)
          | Some(KStoptr stloc) ⇒ (case t of (type.Storage struct) ⇒ SCon struct stloc (sto (Address
ev)) | _ ⇒ False)
          | _ ⇒ False)
        | (Storeloc loc) ⇒ (case t of (type.Storage typ) ⇒ SCon typ loc (sto (Address ev)) | _ ⇒
False))"
    using assms(1) unfolding TypeSafe_def typeCompat_def by simp
  then have "case l2 of (Stackloc loc) ⇒ ∃v. accessStore loc sck = Some v"
    using assms(2) assms(3) denvalue.simps(5) case_optionE by fastforce
  then show ?thesis using assms(3) by simp
qed
```

7.8 *astack* preserves unique locations

```
lemma updateEnvUniqueLocs:
  assumes "TypeSafe ev' (Accounts st) sck' mem' (Storage st) cd'"
    and "(k',e) = astack ip (v) (p) (sck', ev')"
  shows "unique_locations (Denvalue e)" unfolding unique_locations_def
proof(intros)
  have a10:"e = (updateEnv ip (v) (Stackloc (ShowL_nat (Toploc sck'))) ev')" using assms(2) by simp
  then have a15:"Denvalue e = Denvalue(ev' (| Denvalue := fmupd ip ((v),(Stackloc (ShowL_nat (Toploc
sck')))) (Denvalue ev') |))" by simp
  then have a17:"(Denvalue e) $$ ip = Some ((v),(Stackloc (ShowL_nat (Toploc sck'))))" by simp
  have a20:"unique_locations (Denvalue ev')" using assms(1) TypeSafe_def by simp
  have a30:"lessThanTopLocs sck'" using assms(1) TypeSafe_def by simp
  fix x' y
  assume aULoc:"x' |∈| fmran (Denvalue e) ∧ y |∈| fmran (Denvalue e) ∧ snd x' = snd y"
  then have a40:" (∀tloc loc. Toploc sck' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
      accessStore loc sck' = None)" using lessThanTopLocs_def Read_Show_nat'_id a30 by auto
  then have "∀x y. ¬((Denvalue ev') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc sck'))))"
using TypeSafe_def assms typeSafeAllStacklocsExist fmranI a40
    by (metis LSubPrefL2_def nle_le option.distinct(1) prod.exhaust_sel)
  then have a50:"∀x' y. ((Denvalue e) $$ x' = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc sck'))))
      ⟶ x' = ip ∧ (fst y) = (v)" using a30 a15 fmranI by auto
  show "x' = y"
  proof(cases "snd x' = (Stackloc (ShowL_nat (Toploc sck')))")
    case True
    then obtain x'' where "(Denvalue e) $$ x'' = Some x'" using fmranI aULoc by auto
    then have "x'' = ip" using a50 True by blast
    then show ?thesis using a50 aULoc True by (metis fmlookup_ran_iff prod_eq_iff)
  next
    case False
    then obtain x'' where a140:"(Denvalue e) $$ x'' = Some x'" using fmranI aULoc by auto
    then have a150:"x'' ≠ ip" using a50 False a17 by (metis option.inject snd_conv)
    have a160:"∀x y. x ≠ ip ∧ (Denvalue e) $$ x = Some y ⟶ (Denvalue ev') $$ x = Some y" using a15
by simp
    then have "(Denvalue ev') $$ x'' = Some x'" using a140 a150 by blast
    then show ?thesis using aULoc a20 unique_locations_def False a160 a17 by (metis fmranE fmranI
option.inject snd_conv)
  qed
qed

lemma sameMemTSafe:
```

```
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "∀struct loc. ((type.Memory struct), (Stackloc loc)) |∈| fmran (Denvalue ev) ∧ accessStore loc
sck = Some (KMemptr stloc) ⟶ MCon struct mem stloc"
  using assms unfolding TypeSafe_def typeCompat_def by fastforce

lemma sameStoLocTSafe:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "∀struct loc. ((type.Storage struct), (Storeloc loc)) |∈| fmran (Denvalue ev) ⟶ SCon struct
loc (sto (Address ev))" using assms
  unfolding TypeSafe_def typeCompat_def by fastforce

lemma typeSafeUnique:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "unique_locations (Denvalue ev)" using assms unfolding TypeSafe_def by simp

lemma typeSafeAccounts:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "balanceTypes acc" using assms unfolding TypeSafe_def by simp

lemma typeSafeEnvAddresses:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "envAddressesWellFormed ev" using assms unfolding TypeSafe_def by simp

lemma typeSafeSvalue:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "svalueTypes (Svalue ev)" using assms unfolding TypeSafe_def by simp

lemma typeSafeCompPointers:
  assumes "TypeSafe ev acc sck mem sto cd"
  shows "compPointers sck (Denvalue ev)" using assms unfolding TypeSafe_def by simp

lemma lexp_stackloc_imps_id:
  assumes "lexp lv env cd st g = Normal ((LStackloc l2, t'), g')"
  shows "∃i. lv = Id i"
proof (cases lv)
  case (Id i)
  then show ?thesis by auto
next
  case (Ref i r)
  then have "lexp (Ref i r) env cd st g =
    (case Denvalue env $$ i of None ⇒ throw Err
     | Some (tp, Stackloc l) ⇒
         (case accessStore l (Stack st) of None ⇒ throw Err
         | Some (KMemptr l') ⇒ (case tp of type.Memory x ⇒ return x | _ ⇒ throw Err) ≫=
             (λt. local.msel True t l' r env cd (st) ≫= (λ(l'', t'). return (LMemloc l'', type.Memory
t')))
         | Some (KStoptr l') ⇒ (case tp of type.Storage x ⇒ return x | _ ⇒ throw Err) ≫=
             (λt. local.ssel t l' r env cd (st) ≫= (λ(l'', t'). return (LStoreloc l'', type.Storage
t')))
         | Some _ ⇒ throw Err)
     | Some (tp, Storeloc l) ⇒ (case tp of type.Storage x ⇒ return x | _ ⇒ throw Err) ≫=
         (λt. local.ssel t l r env cd (st) ≫= (λ(l', t'). return (LStoreloc l', type.Storage t'))))
     g" using lexp.simps(2)[of i r env cd st g] Ref by simp
  then have "¬(∃l tp. lexp (Ref i r) env cd (st) g = Normal ((LStackloc l, tp), g'))"
    by (auto split: option.splits prod.splits denvalue.split_asm type.splits stackvalue.splits
result.splits)
  then have False using assms Ref by simp
  then show ?thesis by simp
qed

lemma lexpDen:
  assumes "lexp lv env cd (st) g = Normal ((LStackloc l2, t'), g')"
    and "lv = Id i"
  shows "(t', Stackloc l2) |∈| fmran (Denvalue env)"
```

proof -
  have *"(case (Denvalue env) $$ i of
      Some (tp, (Stackloc l)) ⇒ return (LStackloc l, tp)
    | Some (tp, (Storeloc l)) ⇒ return (LStoreloc l, tp)
    | _ ⇒ throw Err) g = Normal ((LStackloc l2, t'), g')"* **using** *lexp.simps(1) assms* **by** *auto*
  then have *"(Denvalue env) $$ i = Some (t', Stackloc l2)"* **by** *(simp split: option.split_asm prod.split_asm denvalue.split_asm)*
  then show *?thesis* **using** *Finite_Map.fmranI[of "(Denvalue env)" i "(t', Stackloc l2)"]* **by** *simp*
qed

lemma *lexpStackloc_imps_inDen:*
  assumes *"lexp lv env cd (st) g = Normal ((LStackloc l2, t'), g')"*
  shows *"(t', Stackloc l2) |∈| fmran (Denvalue env)"* **using** *assms lexpDen lexp_stackloc_imps_id* **by** *metis*

lemma *stackLocs_imp_NotDen:*
  assumes *"lessThanTopLocs sck'"*
    and *"TypeSafe ev' (Accounts st) sck' mem' (Storage st) cd'"*
  shows *"∀x y. (Denvalue ev') $$ x = Some y ⟶ snd y ≠ (Stackloc (ShowL$_{nat}$ (Toploc sck')))"*
proof -
  have *" accessStore (ShowL$_{nat}$ (Toploc sck')) sck' = None"* **using** *lessThanTopLocs_def Read_Show_nat'_id assms(1)* **using** *LSubPrefL2_def* **by** *auto*
  then show *?thesis* **using** *assms(2)* **unfolding** *TypeSafe_def typeCompat_def* **using** *fmranI* **by** *fastforce*
qed

lemma *typeSafe_noDenElementOverToploc_mem:*
  assumes *" TypeSafe env (Accounts st) (Stack st) mem (Storage st) cd"*
    and *"((type.Memory struct), (Stackloc loc)) |∈| fmran (Denvalue env)"*
    and *"(accessStore loc (Stack st)) = Some(KMemptr stloc)"*
  shows *"¬LSubPrefL2 stloc (ShowL$_{nat}$ (Toploc mem))"*
proof
  assume **:"LSubPrefL2 stloc (ShowL$_{nat}$ (Toploc mem))"*
  have *h1:"((∀tloc loc. Toploc mem ≤ tloc ∧ LSubPrefL2 loc (ShowL$_{nat}$ (tloc)) ⟶ accessStore loc mem = None) ∧
   (∀loc y. accessStore loc mem = Some y ⟶ (∃tloc<Toploc mem. LSubPrefL2 loc (ShowL$_{nat}$ tloc))))"*
    using *assms(1)* **unfolding** *TypeSafe_def lessThanTopLocs_def* **by** *blast*
  have *h2:"MCon struct mem stloc"* **using** *assms* **unfolding** *TypeSafe_def typeCompat_def* **by** *fastforce*
  then show *False*
    using ** h1 MCon_imps_Some lessThanSome_imps_Locs lessThanSome_imps_Locs_hash lessThanTopLocs_def* **by** *fast*
qed

lemma *typeSafeAllPtrsNotTop:*
  assumes *"TypeSafe ev acc sck mem sto cd"*
  assumes *"(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev)"*
    and *"accessStore l sck = Some (KMemptr ptr)"*
  shows *"∀tl. tl ≥ Toploc mem ⟶
        (¬LSubPrefL2 ptr (ShowL$_{nat}$ tl))∧
        (¬LSubPrefL2 (ShowL$_{nat}$ tl) ptr)"*
proof *intros*
  fix *tl*
  assume **:" Toploc mem ≤ tl"*
  have *ptrMCon:"MCon t mem ptr"* **using** *assms(1,2,3)* **unfolding** *TypeSafe_def typeCompat_def* **by** *force*
  have *SomeA:"∃x i. accessStore ptr mem = Some x ∨ accessStore (hash ptr (ShowL$_{nat}$ i)) mem = Some x"*
    using *MCon_imps_Some[OF ptrMCon]* **by** *blast*
  have *toplocs:"lessThanTopLocs mem"* **using** *assms(1)* **unfolding** *TypeSafe_def* **by** *blast*
  then have *noneA:"accessStore (ShowL$_{nat}$ (Toploc mem)) mem = None
          ∧ (∀locs. accessStore (hash (ShowL$_{nat}$ (Toploc mem)) locs) mem = None)"*
    unfolding *lessThanTopLocs_def*
    using *LSubPrefL2_def* **by** *blast*
  show *"¬ LSubPrefL2 ptr (ShowL$_{nat}$ tl)"*
    using *LSubPrefL2_def SomeA hash_suffixes_associative noneA **
    by *(metis lessThanTopLocs_def option.discI toplocs)*
  show *"¬ LSubPrefL2 (ShowL$_{nat}$ tl) ptr"* **using** *LSubPrefL2_def MemLSubPrefL2_specific_imps_general **

```
    using MemLSubPrefTransitive ‹¬ LSubPrefL2 ptr (ShowL_nat tl)› by blast

qed

lemma typeSafeAllMemPtrsCantTop:
  assumes "TypeSafe ev acc sck mem sto cd"
  assumes "(type.Memory tp, Stackloc l) |∈| fmran (Denvalue ev)"
    and "accessStore l sck = Some (KMemptr ptr)"
  shows "∀ len arr loc tl. tl ≥ Toploc mem ∧ tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem len arr
ptr loc ⟶
        (¬LSubPrefL2 loc (ShowL_nat tl))∧
        (¬LSubPrefL2 (ShowL_nat tl) loc)"
proof intros
  fix len arr loc tl
  assume *:"Toploc mem ≤ tl ∧ tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem len arr ptr loc"
  then have ptrMCon:"MCon (MTArray len arr) mem ptr" using assms(1,2,3) unfolding TypeSafe_def
typeCompat_def by fastforce
  have SomeA:"∃ x i. accessStore ptr mem = Some x ∨ accessStore (hash ptr (ShowL_nat i)) mem = Some x"
    using MCon_imps_Some[OF ptrMCon] by blast
  have toplocs:"lessThanTopLocs mem" using assms(1) unfolding TypeSafe_def by blast
  then have noneA:"accessStore (ShowL_nat (Toploc mem)) mem = None
          ∧ (∀ locs. accessStore (hash (ShowL_nat (Toploc mem)) locs) mem = None)"
    unfolding lessThanTopLocs_def
    using LSubPrefL2_def by blast
  show "¬ LSubPrefL2 loc (ShowL_nat tl)" using * noneA SomeA
    using assms(1,2,3) ptrMCon
    by (metis LSubPrefL2_def MCon_imps_Some hash_flatten_right lessThanTopLocs_def
        mcon_typedptrs_imps_existence option.discI toplocs)
  then show "¬ LSubPrefL2 (ShowL_nat (tl)) loc" using * noneA SomeA LSubPrefL2_def
MemLSubPrefL2_specific_imps_general
    by metis
qed

lemma AccessedMemPtrsCantTop:
  assumes "lessThanTopLocs mem"
  shows "∀ len arr loc tl. tl ≥ Toploc mem ∧ tp = MTArray len arr ∧
        MCon (MTArray len arr) mem ptr ∧ TypedMemSubPrefPtrs mem len arr ptr loc ⟶
        (¬LSubPrefL2 loc (ShowL_nat tl))∧
        (¬LSubPrefL2 (ShowL_nat tl) loc)"
proof intros
  fix len arr loc tl
  assume *:"Toploc mem ≤ tl ∧ tp = MTArray len arr ∧ MCon (MTArray len arr) mem ptr ∧
TypedMemSubPrefPtrs mem len arr ptr loc"
  then have ptrMCon:"MCon (MTArray len arr) mem ptr" by fastforce
  have SomeA:"∃ x i. accessStore ptr mem = Some x ∨ accessStore (hash ptr (ShowL_nat i)) mem = Some x"
    using MCon_imps_Some[OF ptrMCon] by blast
  have toplocs:"lessThanTopLocs mem" using assms(1) unfolding TypeSafe_def by blast
  then have noneA:"accessStore (ShowL_nat (Toploc mem)) mem = None
          ∧ (∀ locs. accessStore (hash (ShowL_nat (Toploc mem)) locs) mem = None)"
    unfolding lessThanTopLocs_def
    using LSubPrefL2_def by blast
  show "¬ LSubPrefL2 loc (ShowL_nat tl)" using * noneA SomeA
    using assms(1) ptrMCon
    by (metis LSubPrefL2_def MCon_imps_Some hash_flatten_right lessThanTopLocs_def
        mcon_typedptrs_imps_existence option.discI toplocs)
  then show "¬ LSubPrefL2 (ShowL_nat (tl)) loc" using * noneA SomeA LSubPrefL2_def
MemLSubPrefL2_specific_imps_general
    by metis
qed

lemma CompMemTypeSubIndexes:
  assumes "CompMemType mem len arr (MTArray x t) pParentPtr p"
    and "subPrefixStructuralConsistency mem"
    and "MCon (MTArray len arr) mem pParentPtr"
```

```
    and "∀i<len. accessTypeStore (hash pParentPtr (ShowL_nat i)) mem = Some arr"
  shows "∀i<x. accessTypeStore (hash p (ShowL_nat i)) mem = Some t" using assms
proof(induction arr arbitrary:len pParentPtr)
  case (MTArray x1 arr)
  obtain i l where idef:"i<len ∧ accessStore (hash pParentPtr (ShowL_nat i)) mem = Some (MPointer l)
            ∧ (l = p ∧ MTArray x1 arr = MTArray x t ∨ CompMemType mem x1 arr (MTArray x t) l p)"
    using MTArray.prems(1) unfolding CompMemType.simps by blast
  then have mc:"MCon (MTArray x1 arr) mem l" using MTArray.prems(3) by auto
  then show ?case
  proof(cases "l = p")
    case True
    then have tps:"MTArray x1 arr = MTArray x t" using idef MTArray.prems
      by (metis existingLocation_imps_allLocs_same)
    have "∀i<len.∃p'. accessStore (hash pParentPtr (ShowL_nat i)) mem = Some (MPointer p')"
      using MTArray.prems(3)
      by (metis MConArrayPointers bot_nat_0.not_eq_extremum less_nat_zero_code)
    then have "∀i<len.∃p'. accessStore (hash pParentPtr (ShowL_nat i)) mem = Some (MPointer p') ∧
        (∃len arr'. MCon (MTArray x1 arr) mem p' ∧ (MTArray x1 arr) = MTArray len arr'
                  ∧ (∀i<len. accessTypeStore (hash p' (ShowL_nat i)) mem = Some arr'))"
      using MTArray.prems(2,4) unfolding subPrefixStructuralConsistency_def by fastforce
    then show ?thesis using idef True tps by auto
  next
    case False
    then have cmp:"CompMemType mem x1 arr (MTArray x t) l p" using idef by simp
    have "∀i<x1. accessTypeStore (hash l (ShowL_nat i)) mem = Some arr"
      using idef MTArray.prems(2,4) unfolding subPrefixStructuralConsistency_def by fastforce
    then have "∀i<x. accessTypeStore (hash p (ShowL_nat i)) mem = Some t"
      using MTArray.IH[OF cmp MTArray.prems(2) mc _] by blast
    then show ?thesis by blast
  qed
next
  case (MTValue x)
  then show ?case by simp
qed


end
```

A storage pointer derived from a declared storage variable is again related to a declared storage variable.

```
lemma CompStoType_ptr_witness:
  assumes decl: "(type.Storage tp0, Storeloc s0) |∈| fmran den"
    and base: "CompStoType tp0 t s0 l'"
    and rel: "t = t' ∧ v = KStoptr l' ∨ extractValueType v ≠ l' ∧ CompStoType t t' l'
(extractValueType v)"
    and ptr: "∃xx. v = KStoptr xx"
  shows "∃stloc tp''. (type.Storage tp'', Storeloc stloc) |∈| fmran den ∧
    (tp'' = t' ∧ v = KStoptr stloc ∨
     extractValueType v ≠ stloc ∧ CompStoType tp'' t' stloc (extractValueType v))"
proof -
  have c: "CompStoType tp0 t' s0 (extractValueType v)"
    using rel base CompStoType_trns by auto
  show ?thesis
  proof (cases "extractValueType v = s0")
    case True
    then have "tp0 = t'" using c CompStoType_sameLoc_sameType by simp
    moreover have "v = KStoptr s0" using ptr True by auto
    ultimately show ?thesis using decl by blast
  next
    case False
    then show ?thesis using decl c by blast
  qed
qed

end
```

8 Storage-to-storage copies (TypeSafe_Copies)

theory *TypeSafe_Copies*
imports *TypeSafe_Support TypeSafe_Def*
begin

Locality and consistency properties of *copyRec* and *copy*. A copy changes only sub-locations of its destination and preserves *SCon* at unrelated locations.

context *typesafe_base*
begin

lemma *copyRec_preserves_dest_root:*
shows *"∀t'. (copyRec (hash ls suffixa) (hash ld suffix') t m' = Some v'')⟶ accessStorage t' ld v'' = accessStorage t' ld m'"*
proof *intros*
fix *t'*
assume ***:"copyRec (hash ls suffixa) (hash ld suffix') t m' = Some v''"*
then show *"accessStorage t' ld v'' = accessStorage t' ld m'"*
proof *(induction t arbitrary: m' v'' suffixa suffix' ls)*
case *(STArray x1 t)*
then have *a60:"Some v'' = iter' (λi. copyRec (hash (hash ls suffixa) (ShowL$_{nat}$ i)) (hash (hash ld suffix') (ShowL$_{nat}$ i)) t) m' x1"*
unfolding *copyRec.simps* **by** *simp*
show *?case*
proof *(induction rule: iter'_induct[OF _ _ a60[symmetric]])*
case *(1 v')*
then have *"v' = m'"* **using** *iter'.simps* **by** *simp*
then show *?case* **by** *auto*
next
case *(2 x v'')*
then obtain *v'*
where *a10:"iter' (λi. copyRec (hash (hash ls suffixa) (ShowL$_{nat}$ i)) (hash (hash ld suffix') (ShowL$_{nat}$ i)) t) m' x = Some v'"*
and *a20:"accessStorage t' ld v' = accessStorage t' ld m'"*
and *a30:"copyRec (hash (hash ls suffixa) (ShowL$_{nat}$ x)) (hash (hash ld suffix') (ShowL$_{nat}$ x)) t v' = Some v''"* **by** *blast*
then show *?case* **using** *STArray.IH hash_suffixes_associative* **by** *auto*
qed
next
case *(STMap x1 t)*
then show *?case*
using *copyRec.simps(3)* **by** *auto*
next
case *(STValue x)*
then show *?case* **using** *accessStorage_def hash_suffixes_associative*
using *copyRec.simps(2) hash_inequality* **by** *force*
qed
qed

lemma *copyrec_SubPrefixes:*
"copyRec srcl destl tp a = Some v''' ⟶
(∀destl' t. (¬LSubPrefL2 destl' destl) ⟶ accessStorage t destl' a = accessStorage t destl' v''')"
proof*(induction tp arbitrary:srcl destl a v''')*
case *(STArray x1 t)*
show *?case*
proof *intros*
fix *destl' ta*
assume ***:" copyRec srcl destl (STArray x1 t) a = Some v'''"*

then have *a5:"Some v''' = iter' (λi. copyRec (hash srcl (ShowL$_{nat}$ i)) (hash destl (ShowL$_{nat}$ i)) t) a x1"*
using *** copyRec.simps(1)[of srcl destl x1 t a]* **by** *simp*

show *"¬ LSubPrefL2 destl' destl ⟹ accessStorage ta destl' a = accessStorage ta destl' v'''"*

```
    proof(induction rule: iter'_induct[OF _ _ a5[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t) a x = Some
v' "
          and a20:"(¬ LSubPrefL2 destl' destl ⟶ accessStorage ta destl' a = accessStorage ta destl'
v')"
          and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t v' = Some v''" by
blast

      then show ?case
      proof(cases t)
        case mtr:(STArray x11 x12)
        then show ?thesis using accessStorage_def "2.IH" "2.prems" Not_Sub_More_Specific
          using STArray by force
      next
        case (STValue x2)
        then have "copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STValue x2) v' = Some
v''" using a30 by simp
        then have "(let e = accessStorage x2 (hash srcl (ShowL_nat x)) v' in Some (v'(hash destl
(ShowL_nat x) $$:= e))) = Some v''"
          unfolding copyRec.simps by simp
        then show ?thesis using accessStorage_def
          using "2.IH" "2.prems" STArray Not_Sub_More_Specific by fastforce
      next
        case (STMap x3 x4)
        then show ?thesis using a30 by (simp add: copyRec.simps(3))
      qed
    qed
  qed
next
  case (STValue x)
  show ?case
  proof intros
    fix destl' t
    assume "copyRec srcl destl (STValue x) a = Some v'''" and "¬ LSubPrefL2 destl' destl"
    then show "accessStorage t destl' a = accessStorage t destl' v'''"
      using accessStorage_def LSubPrefL2_def copyRec.simps(2) by auto
  qed
next
  case (STMap x1 tp)
  then show ?case using copyRec.simps(3) by auto
qed

lemma copyrec_SubPrefixes_typedPref:
  "copyRec srcl destl tp a = Some v''' ⟶
  (∀ destl' t. ¬ TypedStoSubpref destl' destl tp ⟶ accessStorage t destl' a = accessStorage t destl'
v''')"
proof(induction tp arbitrary:srcl destl a v''' )
  case (STArray x1 tp)
  show ?case
  proof intros
    fix destl' t
    assume **:"copyRec srcl destl (STArray x1 tp) a = Some v'''"
      and ***:"¬ TypedStoSubpref destl' destl (STArray x1 tp)"
    have a5:"Some v''' = iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp) a
x1"
      using ** unfolding copyRec.simps by simp

    show "accessStorage t destl' a = accessStorage t destl' v'''" using ***
    proof(induction rule: iter'_induct[OF _ _ a5[symmetric]])
```

```
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp) a x = Some
v'"
          and a20:"(¬ TypedStoSubpref destl' destl (STArray x tp) ⟶ accessStorage t destl' a =
accessStorage t destl' v')"
          and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) tp v' = Some v''" by
blast
      then show ?case
      proof(cases tp)
        case (STArray x11 x12)
        then show ?thesis
          using "2.prems" STArray.IH a20 a30 by force
      next
        case (STMap x21 x22)
        then show ?thesis
          using a30 copyRec.simps(3) by auto
      next
        case (STValue x3)
        then show ?thesis
          using "2.prems" STArray a20 a30 by auto
      qed
    qed
  qed
next
  case (STMap x1 tp)
  then show ?case
    by (simp add: copyRec.simps(3))
next
  case (STValue x)
  show ?case
  proof intros
    fix destl' t
    assume *:"copyRec srcl destl (STValue x) a = Some v''''"
      and **:"¬ TypedStoSubpref destl' destl (STValue x)"
    then show " accessStorage t destl' a = accessStorage t destl' v''''" using accessStorage_def
      using copyRec.simps(2) by auto
  qed
qed

lemma copyrec_SubPrefixes_typedPref_rev:
  "copyRec srcl destl tp a = Some v''' ⟶
  (∀ destl' t. accessStorage t destl' a ≠ accessStorage t destl' v''' ⟶ TypedStoSubpref destl' destl
tp)"
proof(induction tp arbitrary:srcl destl a v''' )
  case (STArray x1 tp)
  show ?case
  proof intros
    fix destl' t
    assume **:"copyRec srcl destl (STArray x1 tp) a = Some v''''"
      and ***:"accessStorage t destl' a ≠ accessStorage t destl' v''' "
    have a5:"Some v''' = iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp) a
x1"
      using ** unfolding copyRec.simps by simp

    show " TypedStoSubpref destl' destl (STArray x1 tp)" using ***
    proof(induction rule: iter'_induct[OF _ _ a5[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
```

```
      then obtain v'
        where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp) a x = Some
v'"
          and a20:"(accessStorage t destl' a ≠ accessStorage t destl' v' ⟶ TypedStoSubpref destl'
destl (STArray x tp))"
          and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) tp v' = Some v''" by
blast
      then show ?case
      proof(cases tp)
        case (STArray x11 x12)
        then show ?thesis
          using "2.prems" STArray.IH a20 a30
          by (metis less_Suc_eq TypedStoSubpref.simps(2))
      next
        case (STMap x21 x22)
        then show ?thesis
          using a30 copyRec.simps(3) by auto
      next
        case (STValue x3)
        then show ?thesis
          using "2.prems" STArray a20 a30
          using less_Suc_eq by fastforce
      qed
    qed
  qed
next
  case (STMap x1 tp)
  then show ?case
    by (simp add: copyRec.simps(3))
next
  case (STValue x)
  show ?case
  proof intros
    fix destl' t
    assume *:"copyRec srcl destl (STValue x) a = Some v'''"
      and **:" accessStorage t destl' a ≠ accessStorage t destl' v''' "
    then show "TypedStoSubpref destl' destl (STValue x)" using accessStorage_def
      using copyRec.simps(2)
      using copyrec_SubPrefixes_typedPref by blast
  qed
qed

lemma copySingleChange:
  assumes "iter' (λi m. copyRec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t m) md x = Some updM"
  shows "∀t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t'
locs updM"
proof (rule iter'_invariant[OF assms(1)])
  show "∀t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t'
locs md" by simp
next
  fix i m m'
  assume IH:"∀t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage
t' locs m"
    and step:"copyRec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t m = Some m'"
  show "∀t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t'
locs m'"
  proof intros
    fix t' locs
    assume *:"¬LSubPrefL2 locs ld ∨ locs = ld"
    then have "accessStorage t' locs md = accessStorage t' locs m" using IH by simp
    moreover have "accessStorage t' locs m = accessStorage t' locs m'"
      using copyrec_SubPrefixes[of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t m m']
      using step * Not_Sub_More_Specific by (metis copyRec_preserves_dest_root)
    ultimately show "accessStorage t' locs md = accessStorage t' locs m'" by simp
```

```
  qed
qed

lemma copySingleChange_typedPref:
  assumes "iter' (λi m. copyRec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t m) md x = Some updM"
  shows "∀ locs t''. ¬TypedStoSubpref locs ld (STArray x t) ⟶ accessStorage t'' locs md =
accessStorage t'' locs updM"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. copyRec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t) md x = Some v'"
      and a20:"(∀ locs t''. ¬ TypedStoSubpref locs ld (STArray x t) ⟶ accessStorage t'' locs md =
accessStorage t'' locs v')"
      and a30:"copyRec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t v' = Some v''" by blast
  show ?case
  proof intros
    fix locs t''
    assume *:"¬ TypedStoSubpref locs ld (STArray (Suc x) t) "
    then show "accessStorage t'' locs md = accessStorage t'' locs v''"
    proof(cases "x = 0")
      case True
      then have "accessStorage t'' locs md = accessStorage t'' locs v'" using a10 by simp
      then show ?thesis using copyrec_SubPrefixes_typedPref a30
        using "*" by simp
    next
      case False
      then show ?thesis using a30 copyrec_SubPrefixes_typedPref *
        by (metis "2.hyps" copyRec.simps(1))
    qed
  qed
qed

lemma copyRec_nonZero_Arrays:
  assumes "copyRec srcl destl t' srcMem = Some v''''"
    and "subStoTp (STArray len t) t'"
    and "len = 0"
  shows "srcMem = v''''" using assms
proof(induction t' arbitrary:srcl destl srcMem v''' len t)
  case (STArray x1 t')
  have a5:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x1 = Some
v''''"
    using STArray.prems(1) unfolding copyRec.simps by simp
  show ?case using STArray.prems(2)
  proof(induction rule: iter'_induct[OF _ _ a5])
    case (1 v')
    then show ?case by auto
  next
    case (2 x v'')
    then obtain v'
      where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
        and a20:"(subStoTp (STArray len t) (STArray x t') ⟶ srcMem = v')"
        and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast
    then show ?case
      using "2.prems" STArray.IH STArray.prems(3) by force
  qed
next
  case (STMap x1 t')
  then show ?case
    by (simp add: copyRec.simps(3))
next
```

```
  case (STValue x)
  then show ?case by simp
qed

lemma iter'Copy_nonZero_Arrays:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
    and "subStoTp (STArray len t) t'"
    and "len = 0"
  shows "srcMem = v'" using assms
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by auto
next
  case (2 x v'')
  then show ?case
    using copyRec_nonZero_Arrays by blast
qed

lemma copyRec_subT_mustNotBeArray:
  assumes "copyRec srcl destl t' srcMem = Some v'''"
    and "∀ len t. subStoTp (STArray len t) t' ⟶ len > 0"
  shows "∀ k v. ¬subStoTp (STMap k v) t'" using assms
proof(induction t' arbitrary:srcl destl srcMem v''')
  case (STArray x1 t')
  have a5:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x1 = Some
v''''
    using STArray.prems(1) unfolding copyRec.simps by simp
  show ?case using STArray.prems(2)
  proof(induction rule: iter'_induct[OF _ _ a5])
    case (1 v')
    then show ?case by auto
  next
    case (2 x v'')
    then obtain v'
      where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
       and a20:"((∀ len t. subStoTp (STArray len t) (STArray x t') ⟶ 0 < len) ⟶ (∀ k v. ¬ subStoTp
(STMap k v) (STArray x t'))) "
       and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast
    then show ?case
      by (metis STArray.IH STArray.prems(2) stypes.distinct(1) subStoTp.simps(2))
  qed
next
  case (STMap x1 t')
  then show ?case
    by (simp add: copyRec.simps(3))
next
  case (STValue x)
  then show ?case by simp
qed

lemma copy_unchanged:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
    and "SCon (STArray x t') srcl srcMem"
    and "SCon (STArray x t') destl srcMem"
  shows "∀ destl' t i' . i'≥x ∧ LSubPrefL2 destl' (hash destl (ShowL_nat i')) ⟶
     accessStorage t destl' srcMem = accessStorage t destl' v'" using assms(2,3)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by auto
next
  case (2 x v'')
```

```
  then obtain v'
    where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
      and a20:"(SCon (STArray x t') srcl srcMem ⟶
          SCon (STArray x t') destl srcMem ⟶ (∀destl' t i'. x ≤ i' ∧ LSubPrefL2 destl' (hash destl
(ShowL_nat i'))
          ⟶ accessStorage t destl' srcMem = accessStorage t destl' v'))"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

  show ?case
  proof intros
    fix destl' t i'
    assume as1:"Suc x ≤ i' ∧ LSubPrefL2 destl' (hash destl (ShowL_nat i'))"
    then have g1:"accessStorage t destl' srcMem = accessStorage t destl' v'" using a20 2 by (metis
Suc_leD subSCon)
    have g5:"(∀destl' t. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStorage t destl' v' =
accessStorage t destl' v'')"
      using a30 copyrec_SubPrefixes[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t' v' v'']
by simp
    have "¬ LSubPrefL2 destl' (hash destl (ShowL_nat x))" using as1
      by (metis LSubPrefL2_index_unique Suc_n_not_le_n)
    then have g10:" accessStorage t destl' v' = accessStorage t destl' v''" using g5 by simp
    then show "accessStorage t destl' srcMem = accessStorage t destl' v''" using g1 by simp
  qed
qed

lemma copy_unchanged_limit:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
  shows "∀destl' . ¬TypedStoSubpref destl' destl (STArray (Suc x) t') ⟶
                   (∀t. accessStorage t destl' srcMem = accessStorage t destl' v')"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by auto
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
      and a20:" (∀destl'. ¬ TypedStoSubpref destl' destl (STArray (Suc x) t') ⟶ (∀t. accessStorage t
destl' srcMem = accessStorage t destl' v'))"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

  show ?case
  proof intros
    fix destl' t
    assume as1:"¬ TypedStoSubpref destl' destl (STArray (Suc (Suc x)) t')"
    then have g1:"accessStorage t destl' srcMem = accessStorage t destl' v'" using a20 2 by simp
    have "(∀destl' t. ¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ accessStorage t destl'
v' = accessStorage t destl' v'')"
      using a30 as1 copyrec_SubPrefixes_typedPref[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t' v' v''] by simp
    then have g10:" accessStorage t destl' v' = accessStorage t destl' v''" using as1 by simp
    then show "accessStorage t destl' srcMem = accessStorage t destl' v''" using g1 by simp
  qed
qed

lemma copy_unchanged_limit_len:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
  shows "∀destl' . ¬TypedStoSubpref destl' destl (STArray (x) t') ⟶
                   (∀t. accessStorage t destl' srcMem = accessStorage t destl' v')"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
```

```
  then show ?case by auto
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
      and a20:" (∀ destl'. ¬ TypedStoSubpref destl' destl (STArray (x) t') ⟶ (∀ t. accessStorage t
destl' srcMem = accessStorage t destl' v'))"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

  show ?case
  proof intros
    fix destl' t
    assume as1:"¬ TypedStoSubpref destl' destl (STArray (Suc ( x)) t')"
    then have g1:"accessStorage t destl' srcMem = accessStorage t destl' v'" using a20 2 by simp
    have "(∀ destl' t. ¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ accessStorage t destl'
v' = accessStorage t destl' v'')"
      using a30 as1 copyrec_SubPrefixes_typedPref[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t' v' v''] by simp
    then have g10:" accessStorage t destl' v' = accessStorage t destl' v''" using as1 by simp
    then show "accessStorage t destl' srcMem = accessStorage t destl' v''" using g1 by simp
  qed
qed

lemma copyRec_unchanged_limit:
  assumes "copyRec srcl destl t' srcMem = Some v'"
  shows "∀ destl'. ¬TypedStoSubpref destl' destl t' ⟶
                   (∀ t. accessStorage t destl' srcMem = accessStorage t destl' v')" using assms
proof(induction t' arbitrary:srcl destl srcMem v')
  case (STArray x1 t')
  have a5:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x1 = Some
v'"
    using STArray.prems(1) unfolding copyRec.simps by simp
  show ?case
  proof(induction rule: iter'_induct[OF _ _ a5])
    case (1 v')
    then show ?case by auto
  next
    case (2 x v'')
    then obtain v'
      where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
        and a20:"(∀ destl'. ¬ TypedStoSubpref destl' destl (STArray x t') ⟶ (∀ t. accessStorage t
destl' srcMem = accessStorage t destl' v'))"
        and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast
    then show ?case
      by (metis "2.hyps" copyRec.simps(1) copyrec_SubPrefixes_typedPref_rev)
  qed
next
  case (STMap x1 t')
  then show ?case
    using copyrec_SubPrefixes_typedPref_rev by blast
next
  case (STValue x)
  then show ?case
    by (simp add: copyrec_SubPrefixes_typedPref)
qed

lemma copy_unchanged_limit_last_index:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
  shows "∀ destl' . TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶
                   (∀ t. accessStorage t destl' srcMem = accessStorage t destl' v')"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
```

```
  case (1 v')
  then show ?case by auto
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x =
Some v'"
      and a20:"(∀ destl'. TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ (∀ t. accessStorage t
destl' srcMem = accessStorage t destl' v'))"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

  show ?case
  proof intros
    fix destl' t
    assume as1:"TypedStoSubpref destl' (hash destl (ShowL_nat (Suc x))) t'"
    then have g1:"accessStorage t destl' srcMem = accessStorage t destl' v'" using a20 2
      by (meson NotRelatedPrnt_imps_notRelatedChild Sto_divergence_imps_notsubloc
TypedStoSubpref_hashes a10 copy_unchanged_limit)
    have "(∀ destl' t. ¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ accessStorage t destl'
v' = accessStorage t destl' v'')"
      using a30 as1 copyrec_SubPrefixes_typedPref[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t' v' v''] by simp
    then have "accessStorage t destl' v' = accessStorage t destl' v''" using as1
      by (metis "2.hyps" NotRelatedPrnt_imps_notRelatedChild Sto_divergence_imps_notsubloc
TypedStoSubpref_hashes copyRec.simps(1) copyrec_SubPrefixes_typedPref g1)
    then show "accessStorage t destl' srcMem = accessStorage t destl' v''" using g1 by simp
  qed
qed

lemma copyRec_notSub_SCon:
  assumes "copyRec srcl destl t' srcMem = Some v'"
    and "SCon t' srcl srcMem"
    and "¬TypedStoSubpref destl srcl t'"
    and "¬TypedStoSubpref srcl destl t'"
    and "SCon t' destl srcMem"
    and "∀ subL. TypedStoSubpref subL srcl t' ⟶ ¬ TypedStoSubpref subL destl t'"
    and "∀ subL. TypedStoSubpref subL destl t' ⟶ ¬ TypedStoSubpref subL srcl t'"
  shows "SCon t' destl v'" using assms
proof(induction t' arbitrary:srcl destl srcMem v')
  case (STArray x1 t')
  have a5:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x1 = Some
v'"
    using STArray.prems(1) unfolding copyRec.simps by simp
  show ?case using STArray.prems(2,3,4,5,6,7)
  proof(induction rule: iter'_induct[OF _ _ a5])
    case (1 v')
    then show ?case by simp
  next
    case (2 x v'')
    then obtain v' where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i))
t') srcMem x = Some v'"
      and a20:" (SCon (STArray x t') srcl srcMem ⟶ ¬ TypedStoSubpref destl srcl (STArray x t')
           ⟶ ¬ TypedStoSubpref srcl destl (STArray x t') ⟶ SCon (STArray x t') destl srcMem
           ⟶ (∀ subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL destl
(STArray x t')) ⟶
             (∀ subL. TypedStoSubpref subL destl (STArray x t') ⟶ ¬ TypedStoSubpref subL srcl
(STArray x t'))
             ⟶ SCon (STArray x t') destl v' )"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

    moreover have p1:"(∀ subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL
destl (STArray x t'))"
      using TypedStoSubpref_arr_longer 2(7) by blast
    moreover have "(∀ subL. TypedStoSubpref subL destl (STArray x t') ⟶ ¬ TypedStoSubpref subL srcl
```

```
(STArray x t'))"
      using TypedStoSubpref_arr_longer 2(8) by blast
    ultimately have a40:"SCon (STArray x t') destl v'" using 2
      using TypedStoSubpref_sameLoc subSCon by presburger

    have g11:"∀ destl'. ¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ (∀ t. accessStorage t
destl' v' = accessStorage t destl' v'')"
      using copyRec_unchanged_limit[OF a30] by blast
    have g12:"∀ destl'. TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ (∀ t. accessStorage t
destl' srcMem = accessStorage t destl' v')"
      using copy_unchanged_limit_last_index[OF a10] by simp
    have g13:"∀ destl'. ¬ TypedStoSubpref destl' destl (STArray x t') ⟶ (∀ t. accessStorage t destl'
srcMem = accessStorage t destl' v')"
      using copy_unchanged_limit_len[OF a10] by blast

    have scSrc:"SCon (STArray (Suc x) t') srcl v'" using g13 p1 "2.prems"(5) 2(3)
SCon_preserved_disjoint_change
      by (meson NotRelatedPrnt_imps_notRelatedChild SCon_NoChange TypedStoSubpref_sameLoc)

    show ?case
    proof(cases "∀ len t. subStoTp (STArray len t) t' ⟶ 0 < len")
      case True
      have a45:"SCon (STArray x t') destl v''" unfolding SCon.simps
      proof intros
        fix i assume a1:"i<x"
        then have a41:"SCon t' (hash destl (ShowL_nat i)) v'" using a40 by simp
        then have g10:" ∀ k v. ¬ subStoTp (STMap k v) t'"
          using copyRec_subT_mustNotBeArray a30 True by simp
        then have g12:"¬ TypedStoSubpref (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x)) t'" using
a1
          by (meson NotRelatedPrnt_imps_notRelatedChild Sto_divergence_imps_notsubloc
TypedStoSubpref_hashes TypedStoSubpref.simps(2))
        then have "∀ l. ¬ TypedStoSubpref (hash (hash destl (ShowL_nat i)) l) (hash destl (ShowL_nat x))
t'"
          by (metis (full_types) TypedStoSubpref_b a1 hash_suffixes_associative hashesInts nat_neq_iff)
        then show "SCon t' (hash destl (ShowL_nat i)) v''" using g11 a41 g10 g12
        proof(induction t' arbitrary:destl i)
          case (STArray x1 t')
          then show ?case
            using NoSubChanged_SCon_Preserved by presburger
        next
          case (STMap x1 t')
          then show ?case by auto
        next
          case (STValue x')
          then show ?case by simp
        qed
      qed

      show ?thesis unfolding SCon.simps
      proof intros
        fix i
        assume as1:"i<Suc x"
        show "SCon t' (hash destl (ShowL_nat i)) v'' "
        proof(cases "i = x")
          case True
          have "SCon t' (hash destl (ShowL_nat x)) srcMem" using 2(6) by simp
          then have cc1:"SCon t' (hash destl (ShowL_nat x)) v'" using g12 SCon_NoChange by simp
          have cc2:"SCon t' (hash srcl (ShowL_nat x)) v'" using scSrc by simp
          have cc3:"¬TypedStoSubpref (hash destl (ShowL_nat x)) (hash srcl (ShowL_nat x)) t'" using 2
            by (metis True TypedStoSubpref_specific_unreachanble_arry as1 TypedStoSubpref.simps(2))
          have cc4:"¬TypedStoSubpref (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t'" using 2
            by (metis True TypedStoSubpref_specific_unreachanble_arry as1 TypedStoSubpref.simps(2))
          have cc5:"∀ subL. TypedStoSubpref subL (hash srcl (ShowL_nat x)) t' ⟶ ¬
```

```
            TypedStoSubpref subL (hash destl (ShowL_nat x)) t'"
          using 2 NotRelatedPrnt_imps_notRelatedChild cc3 cc4 by blast
        have cc6:"∀subL. TypedStoSubpref subL (hash destl (ShowL_nat x)) t' ⟶ ¬
            TypedStoSubpref subL (hash srcl (ShowL_nat x)) t'"
          using 2 NotRelatedPrnt_imps_notRelatedChild cc3 cc4 by blast
        then show ?thesis using STArray.IH[OF a30 cc2 cc3 cc4 cc1 cc5 cc6] True by blast
      next
        case False
        then have "i<x" using as1 by simp
        then show ?thesis using a45 by simp
      qed
    qed
  next
    case False
    then have a9:"∃len t. subStoTp (STArray len t) t' ∧ 0 = len" by simp
    then have "v' = v''" using copyRec_nonZero_Arrays a30 by blast
    moreover have "srcMem = v'" using iter'Copy_nonZero_Arrays[OF a10] a9 by auto
    ultimately show ?thesis using a40 2(6) by simp
  qed
qed
next
  case (STMap x1 t')
  then show ?case
    by (metis copyRec.simps(3) option.distinct(1))
next
  case (STValue x)
  have "(let e = accessStorage x srcl srcMem in Some (srcMem(destl $$:= e))) = Some v'" using
STValue(1) unfolding copyRec.simps by simp
  then have "(accessStorage x destl v') = (accessStorage x srcl srcMem)" unfolding accessStorage_def
by auto
  moreover have "typeCon x (accessStorage x srcl srcMem)" using STValue(2) unfolding SCon.simps by
simp
  ultimately show ?case unfolding SCon.simps by simp
qed

lemma copy_notSub_SCon:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t') srcMem x = Some
v'"
    and "SCon (STArray x t') srcl srcMem"
    and "¬TypedStoSubpref destl srcl (STArray x t')"
    and "¬TypedStoSubpref srcl destl (STArray x t')"
    and "SCon (STArray x t') destl srcMem"
    and "∀subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL destl (STArray x
t')"
    and "∀subL. TypedStoSubpref subL destl (STArray x t') ⟶ ¬ TypedStoSubpref subL srcl (STArray x
t')"
  shows "SCon (STArray x t') destl v'" using assms(2,3,4,5,6,7)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v' where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t')
srcMem x = Some v'"
    and a20:" (SCon (STArray x t') srcl srcMem ⟶ ¬ TypedStoSubpref destl srcl (STArray x t') ⟶ ¬
        TypedStoSubpref srcl destl (STArray x t') ⟶ SCon (STArray x t') destl srcMem
        ⟶ (∀subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL destl (STArray
x t')) ⟶
          (∀subL. TypedStoSubpref subL destl (STArray x t') ⟶ ¬ TypedStoSubpref subL srcl (STArray x
t'))
          ⟶ SCon (STArray x t') destl v' )"
    and a30:"copyRec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' v' = Some v''" by blast

  moreover have p1:"(∀subL. TypedStoSubpref subL srcl (STArray x t') ⟶ ¬ TypedStoSubpref subL destl
```

```
(STArray x t'))"
    using TypedStoSubpref_arr_longer 2(7) by blast
  moreover have "(∀subL. TypedStoSubpref subL destl (STArray x t') ⟶ ¬ TypedStoSubpref subL srcl
(STArray x t'))"
    using TypedStoSubpref_arr_longer 2(8) by blast
  ultimately have a40:"SCon (STArray x t') destl v'" using 2
    using TypedStoSubpref_sameLoc subSCon by presburger

  have g11:"∀destl'. ¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ (∀t. accessStorage t
destl' v' = accessStorage t destl' v'')"
    using copyRec_unchanged_limit[OF a30] by blast
  have g12:"∀destl'. TypedStoSubpref destl' (hash destl (ShowL_nat x)) t' ⟶ (∀t. accessStorage t
destl' srcMem = accessStorage t destl' v')"
    using copy_unchanged_limit_last_index[OF a10] by simp
  have g13:"∀destl'. ¬ TypedStoSubpref destl' destl (STArray x t') ⟶ (∀t. accessStorage t destl'
srcMem = accessStorage t destl' v')"
    using copy_unchanged_limit_len[OF a10] by blast

  have scSrc:"SCon (STArray (Suc x) t') srcl v'" using g13 p1 "2.prems"(5) 2(3)
SCon_preserved_disjoint_change
    by (meson NotRelatedPrnt_imps_notRelatedChild SCon_NoChange TypedStoSubpref_sameLoc)

  show ?case
  proof(cases "∀len t. subStoTp (STArray len t) t' ⟶ 0 < len")
    case True
    have a45:"SCon (STArray x t') destl v''" unfolding SCon.simps
    proof intros
      fix i assume a1:"i<x"
      then have a41:"SCon t' (hash destl (ShowL_nat i)) v'" using a40 by simp
      then have g10:" ∀k v. ¬ subStoTp (STMap k v) t'"
        using copyRec_subT_mustNotBeArray a30 True by simp
      then have g12:"¬ TypedStoSubpref (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x)) t'" using
a1
        by (meson NotRelatedPrnt_imps_notRelatedChild Sto_divergence_imps_notsubloc
TypedStoSubpref_hashes TypedStoSubpref.simps(2))
      then have "∀l. ¬ TypedStoSubpref (hash (hash destl (ShowL_nat i)) l) (hash destl (ShowL_nat x))
t'"
        by (metis (full_types) TypedStoSubpref_b a1 hash_suffixes_associative hashesInts nat_neq_iff)
      then show "SCon t' (hash destl (ShowL_nat i)) v''" using g11 a41 g10 g12
      proof(induction t' arbitrary:destl i)
        case (STArray x1 t')
        then show ?case
          using NoSubChanged_SCon_Preserved by presburger
      next
        case (STMap x1 t')
        then show ?case by auto
      next
        case (STValue x')
        then show ?case by simp
      qed
    qed

    show ?thesis unfolding SCon.simps
    proof intros
      fix i
      assume as1:"i<Suc x"
      show "SCon t' (hash destl (ShowL_nat i)) v'' "
      proof(cases "i = x")
        case True
        have "SCon t' (hash destl (ShowL_nat x)) srcMem" using 2(6) by simp
        then have cc1:"SCon t' (hash destl (ShowL_nat x)) v'" using g12 SCon_NoChange by simp
        have cc2:"SCon t' (hash srcl (ShowL_nat x)) v'" using scSrc by simp
        have cc3:"¬TypedStoSubpref (hash destl (ShowL_nat x)) (hash srcl (ShowL_nat x)) t'" using 2
          by (metis True TypedStoSubpref_specific_unreachanble_arry as1 TypedStoSubpref.simps(2))
```

```
      have cc4:"¬TypedStoSubpref (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t'" using 2
        by (metis True TypedStoSubpref_specific_unreachanble_arry as1 TypedStoSubpref.simps(2))
      have cc5:"∀subL. TypedStoSubpref subL (hash srcl (ShowL_nat x)) t' ⟶ ¬ TypedStoSubpref subL
(hash destl (ShowL_nat x)) t'"
        using 2 NotRelatedPrnt_imps_notRelatedChild cc3 cc4 by blast
      have cc6:"∀subL. TypedStoSubpref subL (hash destl (ShowL_nat x)) t' ⟶ ¬ TypedStoSubpref subL
(hash srcl (ShowL_nat x)) t'"
        using 2 NotRelatedPrnt_imps_notRelatedChild cc3 cc4 by blast
      then show ?thesis using copyRec_notSub_SCon[OF a30 cc2 cc3 cc4 cc1 cc5 cc6] True by blast
    next
      case False
      then have "i<x" using as1 by simp
      then show ?thesis using a45 by simp
    qed
  qed
next
  case False
  then have a9:"∃len t. subStoTp (STArray len t) t' ∧ 0 = len" by simp
  then have "v' = v''" using copyRec_nonZero_Arrays a30 by blast
  moreover have "srcMem = v'" using iter'Copy_nonZero_Arrays[OF a10] a9 by auto
  ultimately show ?thesis using a40 2(6) by simp
qed
qed

lemma copyrec_same_SCon:
  assumes "SCon t' srcl srcMem"
    and "copyRec srcl srcl t' srcMem = Some v''''"
  shows "SCon t' srcl v''''" using assms
proof (induction t' arbitrary:srcl srcMem v''' )
  case (STArray x1 t')
  have a5:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash srcl (ShowL_nat i)) t') srcMem x1 = Some
v''''"
    using STArray.prems(2) unfolding copyRec.simps by simp
  show ?case using STArray.prems(1)
  proof(induction rule: iter'_induct[OF _ _ a5])
    case (1 v')
    then show ?case by simp
  next
    case (2 x v'')
    then obtain v' where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash srcl (ShowL_nat i)) t')
srcMem x = Some v'"
      and a20:"(SCon (STArray x t') srcl srcMem ⟶ SCon (STArray x t') srcl v')"
      and a30:"copyRec (hash srcl (ShowL_nat x)) (hash srcl (ShowL_nat x)) t' v' = Some v''" by blast
    then have a40:" SCon (STArray x t') srcl v'" using 2 by auto
    have a50:"∀l t. LSubPrefL2 l (hash srcl (ShowL_nat x)) ⟶ accessStorage t l srcMem = accessStorage
t l v'" using a10
      using "2.prems" copy_unchanged subSCon by blast
    then show ?case
    proof(cases "∀len t. subStoTp (STArray len t) t' ⟶ 0 < len")
      case t2:True
      then have g10:" ∀k v. ¬ subStoTp (STMap k v) t'"
        using copyRec_subT_mustNotBeArray a30 by blast
      then have "SCon t' (hash srcl (ShowL_nat x)) v'" using 2(3) a50
      proof(induction t' arbitrary:srcl x)
        case (STArray x1 t')
        have " ∀i<x1. SCon t' (hash (hash srcl (ShowL_nat x)) (ShowL_nat i)) v'"
        proof(intros)
          fix i
          assume asi1:"i<x1"
          then show "SCon t' (hash (hash srcl (ShowL_nat x)) (ShowL_nat i)) v'" using STArray.IH
STArray.prems
            by (meson LSubPrefL2_def NoSubChanged_SCon_Preserved lessI SCon.simps(2))
        qed
        then show ?case unfolding SCon.simps by blast
```

```
      next
        case (STMap x1 t')
        then show ?case by auto
      next
        case (STValue x')
        then show ?case
          by (metis LSubPrefL2_def lessI SCon.simps(1,2))
      qed
      have g11:"∀destl'. ¬ TypedStoSubpref destl' (hash srcl (ShowL_nat x)) t' ⟶ (∀t. accessStorage t
destl' v' = accessStorage t destl' v'')"
        using copyRec_unchanged_limit[OF a30] by blast
      have a55:"SCon (STArray x t') srcl v''" unfolding SCon.simps
      proof intros
        fix i assume a1:"i<x"
        then have a41:"SCon t' (hash srcl (ShowL_nat i)) v'" using a40 by simp
        then have g12:"¬ TypedStoSubpref (hash srcl (ShowL_nat i)) (hash srcl (ShowL_nat x)) t'" using
a1
          by (meson NotRelatedPrnt_imps_notRelatedChild Sto_divergence_imps_notsubloc
TypedStoSubpref_hashes TypedStoSubpref.simps(2))
        then have "∀l. ¬ TypedStoSubpref (hash (hash srcl (ShowL_nat i)) l) (hash srcl (ShowL_nat x))
t'"
          by (metis (full_types) TypedStoSubpref_b a1 hash_suffixes_associative hashesInts nat_neq_iff)
        then show "SCon t' (hash srcl (ShowL_nat i)) v''" using g11 a41 g10 g12
        proof(induction t' arbitrary:srcl i)
          case (STArray x1 t')
          then show ?case
            using NoSubChanged_SCon_Preserved by presburger
        next
          case (STMap x1 t')
          then show ?case by auto
        next
          case (STValue x')
          then show ?case by simp
        qed
      qed

      then have a60:"SCon t' (hash srcl (ShowL_nat x)) v'''
        using STArray.IH[OF _ a30]
        by (simp add: <SCon t' (hash srcl (ShowL_nat x)) v'>)
      show ?thesis unfolding SCon.simps
      proof intros
        fix i assume as1:"i<Suc x"
        show "SCon t' (hash srcl (ShowL_nat i)) v'''
        proof(cases "i ≠ x")
          case True
          then show ?thesis using a55 as1 by auto
        next
          case False
          then show ?thesis using a60 by simp
        qed
      qed
    next
      case False
      then have a9:"∃len t. subStoTp (STArray len t) t' ∧ 0 = len" by simp
      then have "v' = v''" using copyRec_nonZero_Arrays a30 by blast
      moreover have "srcMem = v'" using iter'Copy_nonZero_Arrays[OF a10] a9 by auto
      ultimately show ?thesis using a40 2(3) by blast
    qed
  qed
next
  case (STMap x1 t')
  then show ?case
    by (metis copyRec_nonZero_Arrays copyRec_subT_mustNotBeArray neq0_conv subStoTp.simps(3))
next
```

```
  case (STValue x)
  then have " (let e = accessStorage x srcl srcMem in Some (srcMem(srcl $$:= e))) = Some v'''"
    unfolding copyRec.simps by blast
  then show ?case
    using STValue(1) using accessStorage_def by fastforce
qed

lemma copy_same_SCon:
  assumes "iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash srcl (ShowL_nat i)) t') srcMem x = Some v'"
    and "SCon (STArray x t') srcl srcMem"
  shows "SCon (STArray x t') srcl v'" using assms(2)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v' where a10:"iter' (λi. copyRec (hash srcl (ShowL_nat i)) (hash srcl (ShowL_nat i)) t')
srcMem x = Some v'"
    and a20:"(SCon (STArray x t') srcl srcMem ⟶ SCon (STArray x t') srcl v')"
    and a30:"copyRec (hash srcl (ShowL_nat x)) (hash srcl (ShowL_nat x)) t' v' = Some v''" by blast
  then show ?case using copyrec_same_SCon
    using "2.hyps" "2.prems" copyRec.simps(1) by presburger
qed

end
end
```

9 Memory initialisation (TypeSafe_Memory_Init)

```
theory TypeSafe_Memory_Init
  imports TypeSafe_Support TypeSafe_Storage TypeSafe_Memory
begin
```

Properties of `minitRec` and `minit`: locality of the writes, self-pointing array pointers, `MCon` of the initialised region, and preservation of `Toploc`.

9.1 Locality of initialisation

```
lemma minitRec_SubPrefixes:
  "minitRec destl tp v' = v''' ⟶
  (∀destl'. (¬LSubPrefL2 destl' destl) ⟶ accessStore destl' v' = accessStore destl' v''')"
proof(induction tp arbitrary:destl v' v''' )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTArray x1 t) v' = v'''"

    then have a5:"v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) v' in iter
(λi. minitRec (hash destl (ShowL_nat i)) t) m x1)"
      using ** unfolding minitRec.simps by (simp )
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) v'" by auto
    have a20:"¬ LSubPrefL2 destl' destl ⟹ accessStore destl' v' = accessStore destl' m"
    proof -
      assume "¬ LSubPrefL2 destl' destl"
      then have "¬((∃i. destl' = hash destl i) ∨ destl' = destl)" using LSubPrefL2_def[of destl'
destl] by auto
      then have "(∀i. destl' ≠ hash destl i) ∧ destl' ≠ destl" by simp
      then show "accessStore destl' v' = accessStore destl' m"
        using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
by auto
    qed
```

```
    then have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a5 mdef by
metis
    show "¬ LSubPrefL2 destl' destl ⟹ accessStore destl' v' = accessStore destl' v'''" using a20
    proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v2
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v2"
          and a20:"(¬ LSubPrefL2 destl' destl ⟶
          (¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl' m) ⟶ accessStore
destl' v' = accessStore destl' v2)"
          and a30:" minitRec (hash destl (ShowL_nat x)) t v2 = v'''" by blast
      then have b40:"accessStore destl' v' = accessStore destl' v2" using 2 by blast
      then show ?case
      proof(cases t)
        case mtr:(MTArray x11 x12)
        then have b10:"v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 x12) v2
                              in iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
m x11)"
          using a30 minitRec.simps(1)[of "(hash destl (ShowL_nat x))" x11 x12] by simp

        then have b45:"(∀ destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore destl'
v2 = accessStore destl' v'')"
          using MTArray mtr a30 by blast
        have "¬ LSubPrefL2 destl' (hash destl (ShowL_nat x))" using Not_Sub_More_Specific 2 by blast
        then show ?thesis using b45 b40 by simp
      next
        case (MTValue x2)
        then have "v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v2"
          using minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2] a30 by simp

        then have b45:"(∀ destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore destl'
v2 = accessStore destl' v'')"
          using MTArray MTValue a30 by blast
        have "¬ LSubPrefL2 destl' (hash destl (ShowL_nat x))" using Not_Sub_More_Specific 2 by blast
        then have b50:"accessStore destl' v' = accessStore destl' v'''" using b45
          using b40 by auto
        then show ?thesis using b40 by simp
      qed
    qed
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) v' = v'''"
      and ***:"¬ LSubPrefL2 destl' destl"

    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) v'" using
minitRec.simps(2)[of destl x ] by simp
    then have "¬((∃ i. destl' = hash destl i) ∨ destl' = destl)" using LSubPrefL2_def[of destl' destl]
*** by auto
    then have "(∀ i. destl' ≠ hash destl i) ∧ destl' ≠ destl" by simp
    then show " accessStore destl' v' = accessStore destl' v'''"
      using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
  qed
qed

lemma minitRec_SubPrefixes_typed:
```

```
  "minitRec destl tp v' = v''' ⟶
  (∀ destl'. (¬LSubPrefL2 destl' destl) ⟶ accessTypeStore destl' v' = accessTypeStore destl' v''')"
proof(induction tp arbitrary:destl v' v''' )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTArray x1 t) v' = v'''"

    then have a5:"v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) v' in iter
(λi. minitRec (hash destl (ShowL_nat i)) t) m x1)"
      using ** unfolding minitRec.simps by (simp )
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) v'" by auto
    have a20:"¬ LSubPrefL2 destl' destl ⟹ accessTypeStore destl' v' = accessTypeStore destl' m"
    proof -
      assume "¬ LSubPrefL2 destl' destl"
      then have "¬((∃i. destl' = hash destl i) ∨ destl' = destl)" using LSubPrefL2_def[of destl'
destl] by auto
      then have "(∀i. destl' ≠ hash destl i) ∧ destl' ≠ destl" by simp
      then show "accessTypeStore destl' v' = accessTypeStore destl' m"
        using mdef unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by auto
    qed

    then have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a5 mdef by
metis
    show "¬ LSubPrefL2 destl' destl ⟹ accessTypeStore destl' v' = accessTypeStore destl' v'''" using
a20
    proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v2
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v2"
          and a20:"(¬ LSubPrefL2 destl' destl ⟶
          (¬ LSubPrefL2 destl' destl ⟶ accessTypeStore destl' v' = accessTypeStore destl' m) ⟶
accessTypeStore destl' v' = accessTypeStore destl' v2)"
          and a30:" minitRec (hash destl (ShowL_nat x)) t v2 = v''" by blast
      then have b40:"accessTypeStore destl' v' = accessTypeStore destl' v2" using 2 by blast
      then show ?case
      proof(cases t)
        case mtr:(MTArray x11 x12)
        then have b10:"v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 x12) v2
                              in iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
m x11)"
          using a30 minitRec.simps(1)[of "(hash destl (ShowL_nat x))" x11 x12] by simp

        then have b45:"(∀ destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessTypeStore
destl' v2 = accessTypeStore destl' v'')"
          using MTArray mtr a30 by blast
        have "¬ LSubPrefL2 destl' (hash destl (ShowL_nat x))" using Not_Sub_More_Specific 2 by blast
        then show ?thesis using b45 b40 by simp
      next
        case (MTValue x2)
        then have "v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v2"
          using minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2] a30 by simp

        then have b45:"(∀ destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessTypeStore
destl' v2 = accessTypeStore destl' v'')"
          using MTArray MTValue a30 by blast
        have "¬ LSubPrefL2 destl' (hash destl (ShowL_nat x))" using Not_Sub_More_Specific 2 by blast
        then have b50:"accessTypeStore destl' v' = accessTypeStore destl' v''" using b45
```

```
          using b40 by auto
        then show ?thesis using b40 by simp
      qed
    qed
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) v' = v'''"
      and ***:"¬ LSubPrefL2 destl' destl"

    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) v'" using
minitRec.simps(2)[of destl x ] by simp
    then have "¬((∃ i. destl' = hash destl i) ∨ destl' = destl)" using LSubPrefL2_def[of destl' destl]
*** by auto
    then have "(∀ i. destl' ≠ hash destl i) ∧ destl' ≠ destl" by simp
    then show " accessTypeStore destl' v' = accessTypeStore destl' v'''"
      using mdef unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def
      by (auto split:option.splits memoryvalue.splits)
  qed
qed

lemma minitSingleChange:
  assumes "iter (λi m'' . minitRec (hash ld (ShowL_nat i)) t m'') m x = updM"
  shows "∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs m = accessStore locs updM"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter (λi. minitRec (hash ld (ShowL_nat i)) t) m x = v'"
      and a20:"(∀ t' locs. ¬ LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs m = accessStore locs
v') "
      and a30:"minitRec (hash ld (ShowL_nat x)) t v' = v''" by blast
  show ?case
  proof intros
    fix locs t'
    assume *:"¬ LSubPrefL2 locs ld ∨ locs = ld"
    then have "accessStore locs m = accessStore locs v'" using a20 by simp
    then show "accessStore locs m = accessStore locs v''"
      using minitRec_SubPrefixes a30 * Not_Sub_More_Specific
      by (metis LSubPrefL2_def hash_inequality hash_suffixes_associative)
  qed
qed

lemma minitSingleChange_typed:
  assumes "iter (λi m'' . minitRec (hash ld (ShowL_nat i)) t m'') m x = updM"
  shows "∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs m = accessTypeStore locs
updM"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter (λi. minitRec (hash ld (ShowL_nat i)) t) m x = v'"
      and a20:"(∀ t' locs. ¬ LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs m =
accessTypeStore locs v') "
      and a30:"minitRec (hash ld (ShowL_nat x)) t v' = v''" by blast
  show ?case
```

```
  proof intros
    fix locs t'
    assume *:"¬ LSubPrefL2 locs ld ∨ locs = ld"
    then have "accessTypeStore locs m = accessTypeStore locs v'" using a20 by simp
    then show "accessTypeStore locs m = accessTypeStore locs v''"
      using minitRec_SubPrefixes_typed a30 * Not_Sub_More_Specific
      by (metis LSubPrefL2_def hash_inequality hash_suffixes_associative)
  qed
qed

lemma minitRec_SubPrefixes_typedPref:
  "minitRec destl tp a = v''' ⟶
  (∀destl'. destl' ≠ destl ∧ (¬ TypedMemSubPref destl' destl tp) ⟶ accessStore destl' a =
accessStore destl' v''')"
proof(induction tp arbitrary:destl a v''' )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:"minitRec destl (MTArray x1 t) a = v''''"

    have a5:"(let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a in iter (λi. minitRec
(hash destl (ShowL_nat i)) t) m x1) = v''''"
      using ** unfolding minitRec.simps by simp
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto
    have a20:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t) ⟹ accessStore destl' a
= accessStore destl' m"
    proof -
      assume asm:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t)"
      then have "¬(∃i<x1. TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∨ destl' = hash destl
(ShowL_nat i))"
        using TypedMemSubPref.simps(2)[of destl' destl x1 t] by auto
      then have "(∀i<x1. ¬TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∧ destl' ≠ hash destl
(ShowL_nat i))" by simp
      then show "accessStore destl' a = accessStore destl' m" using mdef asm unfolding
updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def by auto
    qed

    then have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1"
      using a5 mdef by metis
    show "destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t) ⟹ accessStore destl' a =
accessStore destl' v''''" using a20
    proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case using 1 by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x t) ⟶
          (destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x t) ⟶ accessStore destl' a =
accessStore destl' m) ⟶
          accessStore destl' a = accessStore destl' v')"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
      then have b40:"accessStore destl' a = accessStore destl' v'" using 2
        by (metis less_Suc_eq TypedMemSubPref.simps(2))
      then show ?case
      proof(cases t)
        case mtr:(MTArray x11 x12)
        then have b10:"v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 x12) v'
                              in iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12) m
x11)"
          using a30 minitRec.simps(1)[of "(hash destl (ShowL_nat x))" x11 x12] by simp
```

```
        then have b45:"(∀ destl'. destl' ≠ (hash destl (ShowL_nat x)) ∧ ¬ TypedMemSubPref
            destl'(hash destl (ShowL_nat x)) t ⟶ accessStore destl' v' = accessStore destl' v'')"
          using MTArray mtr a30 by blast
        have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using 2
          using Not_Sub_More_Specific_more_speific by blast
        then show ?thesis using b45 b40
          using "2.prems"(1) by force
      next
        case (MTValue x2)
        then have "v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v'"
          using minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2] using a30 by simp
        then have b45:"(∀ destl'. destl' ≠ (hash destl (ShowL_nat x))
                        ∧ ¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t
                        ⟶ accessStore destl' v' = accessStore destl' v'')"
          using MTArray MTValue a30 by blast
        have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using Not_Sub_More_Specific_more_speific
2 by blast
        then have b50:"accessStore destl' v' = accessStore destl' v''" using b45
          by (simp add: MTValue)
        then show ?thesis using b40 by simp
      qed
    qed
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) a = v'''"
    assume ***:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTValue x)"
    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) a"
      using ** minitRec.simps(2)[of destl x] by simp
    assume ***:" destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTValue x)"
    then show "accessStore destl' a = accessStore destl' v'''"
      using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
  qed
qed

lemma minitRec_SubPrefixes_typedPref_typed:
  "minitRec destl tp a = v''' ⟶
  (∀ destl'. destl' ≠ destl ∧ (¬ TypedMemSubPref destl' destl tp) ⟶ accessTypeStore destl' a =
accessTypeStore destl' v''')"
proof(induction tp arbitrary:destl a v''' )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:"minitRec destl (MTArray x1 t) a = v'''"

    have a5:"(let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a in iter (λi. minitRec
(hash destl (ShowL_nat i)) t) m x1) = v'''"
      using ** unfolding minitRec.simps by simp
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto
    have a20:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t) ⟹ accessTypeStore
destl' a = accessTypeStore destl' m"
    proof -
      assume asm:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t)"
      then have "¬(∃ i<x1. TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∨ destl' = hash destl
(ShowL_nat i))"
        using TypedMemSubPref.simps(2)[of destl' destl x1 t] by auto
      then have "(∀ i<x1. ¬TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∧ destl' ≠ hash destl
(ShowL_nat i))" by simp
```

```
      then show "accessTypeStore destl' a = accessTypeStore destl' m"
        using mdef asm unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by auto
    qed

    then have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1"
      using a5 mdef by metis
    show "destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t) ⟹ accessTypeStore destl' a
= accessTypeStore destl' v''''" using a20
    proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case using 1 by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x t) ⟶
          (destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x t) ⟶ accessTypeStore destl' a
= accessTypeStore destl' m) ⟶
          accessTypeStore destl' a = accessTypeStore destl' v')"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
      then have b40:"accessTypeStore destl' a = accessTypeStore destl' v'" using 2
        by (metis less_Suc_eq TypedMemSubPref.simps(2))
      then show ?case
      proof(cases t)
        case mtr:(MTArray x11 x12)
        then have b10:"v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 x12) v'
                              in iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12) m
x11)"
          using a30 minitRec.simps(1)[of "(hash destl (ShowL_nat x))" x11 x12] by simp

        then have b45:"(∀destl'. destl' ≠ (hash destl (ShowL_nat x)) ∧ ¬ TypedMemSubPref destl'(hash
destl (ShowL_nat x)) t
                        ⟶ accessTypeStore destl' v' = accessTypeStore destl' v'')"
          using MTArray mtr a30 by blast
        have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using 2
          using Not_Sub_More_Specific_more_speific by blast
        then show ?thesis using b45 b40
          using "2.prems"(1) by force
      next
        case (MTValue x2)
        then have "v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v'"
          using minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2] using a30 by simp
        then have b45:"(∀destl'. destl' ≠ (hash destl (ShowL_nat x))
                        ∧ ¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t
                        ⟶ accessTypeStore destl' v' = accessTypeStore destl' v'')"
          using MTArray MTValue a30 by blast
        have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using Not_Sub_More_Specific_more_speific
2 by blast
        then have b50:"accessTypeStore destl' v' = accessTypeStore destl' v''" using b45
          by (simp add: MTValue)
        then show ?thesis using b40 by simp
      qed
    qed
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) a = v''''"
    assume ***:"destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTValue x)"
    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) a"
```

```
      using ** minitRec.simps(2)[of destl x] by simp
    assume ***:" destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTValue x)"
    then show "accessTypeStore destl' a = accessTypeStore destl' v'''"
      using mdef unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def
      by (auto split:option.splits memoryvalue.splits)
  qed
qed

lemma minitSingleChange_typedPref:
  assumes "iter (λi m'' . minitRec (hash ld (ShowL_nat i)) t m'') m x = updM"
  shows "∀locs. ¬TypedMemSubPref locs ld (MTArray x t) ∨ locs = ld ⟶ accessStore locs m =
accessStore locs updM"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter (λi. minitRec (hash ld (ShowL_nat i)) t) m x = v'"
      and a20:"(∀locs. ¬ TypedMemSubPref locs ld (MTArray x t) ∨ locs = ld ⟶ accessStore locs m =
accessStore locs v')"
      and a30:"minitRec (hash ld (ShowL_nat x)) t v' = v''" by blast
  show ?case
  proof intros
    fix locs
    assume *:"¬ TypedMemSubPref locs ld (MTArray (Suc x) t) ∨ locs = ld"
    then have "accessStore locs m = accessStore locs v'" using a20 by auto
    then show "accessStore locs m = accessStore locs v''" using minitRec_SubPrefixes_typedPref
      using "*" a30
      by (metis LSubPrefL2_def hash_inequality hash_suffixes_associative lessI minitRec_SubPrefixes
TypedMemSubPref.simps(2))
  qed
qed

lemma minitSingleChange_typedPref_typed:
  assumes "iter (λi m'' . minitRec (hash ld (ShowL_nat i)) t m'') m x = updM"
  shows "∀locs. ¬TypedMemSubPref locs ld (MTArray x t) ∨ locs = ld ⟶ accessTypeStore locs m =
accessTypeStore locs updM"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter (λi. minitRec (hash ld (ShowL_nat i)) t) m x = v'"
      and a20:"(∀locs. ¬ TypedMemSubPref locs ld (MTArray x t) ∨ locs = ld ⟶ accessTypeStore locs m
= accessTypeStore locs v')"
      and a30:"minitRec (hash ld (ShowL_nat x)) t v' = v''" by blast
  show ?case
  proof intros
    fix locs
    assume *:"¬ TypedMemSubPref locs ld (MTArray (Suc x) t) ∨ locs = ld"
    then have "accessTypeStore locs m = accessTypeStore locs v'" using a20 by auto
    then show "accessTypeStore locs m = accessTypeStore locs v''" using minitRec_SubPrefixes_typedPref_typed
      using "*" a30
      by (metis LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality lessI
neg_MemLSubPrefL2_imps_TypedMemSubPref)
  qed
qed
```

9.2 Array roots and self-pointing pointers

```
lemma minitRec_array_root_both:
```

```
  assumes step: "minitRec destl (MTArray x t) m0 = v"
  shows "accessTypeStore destl v = Some (MTArray x t)
    ∧ accessStore destl v = Some (MPointer destl)"
proof -
  let ?m = "updateTypedStore destl (MPointer destl) (MTArray x t) m0"
  have run: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) ?m x = v"
    using step unfolding Let_def minitRec.simps(1)
    by blast
  have keep_type:
    "accessTypeStore destl ?m = accessTypeStore destl v"
    using minitSingleChange_typedPref_typed[OF run] by simp
  have keep_store:
    "accessStore destl ?m = accessStore destl v"
    using minitSingleChange[OF run] by simp
  show ?thesis
    using keep_type keep_store
    unfolding accessTypeStore_def accessStore_def updateTypeStore_def updateTypedStore_def
updateStore_def
    by simp
qed


lemma minitRecPtrsPointToSelf:
  shows "minitRec destl t destm = v'
         ⟶ (∀l l'. (TypedMemSubPref l destl t ∨ l = destl) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)"
proof(induction t arbitrary: destl destm v')
  case (MTArray x1 t)
  show ?case
  proof intros
    fix l l'
    assume *:"minitRec destl (MTArray x1 t) destm = v'"
      and **:"(TypedMemSubPref l destl (MTArray x1 t) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have a10:"v' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm in iter (λi.
minitRec (hash destl (ShowL_nat i)) t) m x1)"
      using * minitRec.simps(1)[of destl x1 t ] by simp
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm" by
auto
    then have a45:"accessStore destl m = Some(MPointer destl)" unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def by simp
    then have a50:"v' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a10 a40 by
presburger
    show "l' = l" using a45 **
    proof(induction rule: iter_ind[OF _ _ a50[symmetric]])
      case (1)
      then show ?case by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(accessStore destl m = Some (MPointer destl) ⟶
          (TypedMemSubPref l destl (MTArray x t) ∨ l = destl) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
      then have "accessStore destl m = Some (MPointer destl)" using 2 by blast
      then show ?case
      proof(cases "l = destl")
        case True
        then show ?thesis using 2(1,3,4)
          by (metis MTArray minitRec_SubPrefixes_typedPref)
      next
        case f1:False
        then show ?thesis
```

```
        proof(cases "TypedMemSubPref l destl (MTArray x t)")
          case True
          have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
          then have b50:"accessStore l v' = accessStore l v''"
            using * a30 by (meson minitRec_SubPrefixes)
          then show ?thesis using a20 2 using True by argo
        next
          case False
          then have b5:"TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
              (ShowL_nat x)" using 2(4) TypedMemSubPref.simps(2)[of l destl "(Suc x)" t] f1
            using less_Suc_eq by auto
          have b10:"accessStore l v'' = Some (MPointer l')" using 2(4) by auto

          have "(∀l l'. (TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
              (ShowL_nat x)) ∧ accessStore l v'' = Some (MPointer l') ⟶ l' = l)"
            using MTArray[of "(hash destl (ShowL_nat x))"] a30 by simp
          then show ?thesis using b5 b10 by blast
        qed
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix l l'
    assume *:"minitRec destl (MTValue x) destm = v'"
      and **:"(TypedMemSubPref l destl (MTValue x) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have "v' = updateTypedStore destl (MValue (ival x)) (MTValue x) destm"
      using minitRec.simps(2)[of destl x ] * by simp
    then obtain v where b10: "accessStore destl v' = Some (MValue v)"
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
    show "l' = l" using b10 ** by auto
  qed
qed

lemma minitSelfPointers:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
  shows "(∀l l'. TypedMemSubPref l destl (MTArray x1 t) ∧ accessStore l v' = Some (MPointer l') ⟶ l'
= l)" using assms(1)
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
      and a20: "(iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v' ⟶
          (∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer l') ⟶ l'
= l))"
      and a30: "minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
  show ?case
  proof intros
    fix l l'
    assume *:"TypedMemSubPref l destl (MTArray (Suc x) t) ∧ accessStore l v'' = Some (MPointer l')"
    show "l' = l"
    proof(cases t)
      case (MTArray x11 x12)

      then show ?thesis
      proof(cases "TypedMemSubPref l destl (MTArray x t)")
```

```
        case True
        have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
        then have b50:"accessStore l v' = accessStore l v''"
          using * minitRec_SubPrefixes a30 by blast
        then show ?thesis using a20 True * b50
          using a10 by presburger
      next
        case False
        then have "TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = (hash destl (ShowL_nat x))"
          using TypedMemSubPref.simps(2)[of l destl "(Suc x)" t] *
          using less_Suc_eq by auto
        then show ?thesis using a30 minitRecPtrsPointToSelf * by blast
      qed
    next
      case (MTValue x2)
      then have b390:"v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2)
v'"
        using a30 minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2 ] by simp
      have "(∃ i<Suc x. TypedMemSubPref l (hash destl (ShowL_nat i)) t ∨ l = hash destl
          (ShowL_nat i))" using TypedMemSubPref.simps(2)[of l destl "Suc x" t] * by auto
      then obtain i where b400:"i< Suc x ∧ TypedMemSubPref l (hash destl (ShowL_nat i)) (MTValue x2)"
using MTValue by auto
      then have "l = (hash destl (ShowL_nat i))" by simp

      then show ?thesis using *
        by (metis "2.IH" TypedMemSubPref.simps(2) less_SucE minitRecPtrsPointToSelf
minitRec_SubPrefixes_typedPref)

    qed
  qed
qed
```

9.3 Consistency of initialised memory

```
lemma minitRecMCon :
  "minitRec destl tp a = v''' ⟶ arraysGreaterZero tp
          ⟶ MCon tp v''' destl"
proof (induction tp arbitrary:destl a v''')
  case (MTArray x1 t)
  show ?case
  proof intros
    assume **:"minitRec destl (MTArray x1 t) a = v''''"
      and ***:"arraysGreaterZero (MTArray x1 t)"
    then have a5:"(let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a in iter (λi.
minitRec (hash destl (ShowL_nat i)) t) m x1) = v''''"
      using ** minitRec.simps(1)[of destl x1 t ] by simp
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto

    then show " MCon (MTArray x1 t) v''' destl"
    proof(cases "x1>0")
      case True
      then have agt:"arraysGreaterZero t"
        using *** unfolding arraysGreaterZero.simps by auto
      have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a5 mdef by metis

      have "∀ i<x1. case accessStore (hash destl (ShowL_nat i)) v''' of None ⇒ False
            | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v''' (hash destl (ShowL_nat i)) | _
⇒ False)
            | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v''' loc2 | MTValue Types
⇒ False)"

      proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
        case (1)
```

```
        then show ?case by simp
      next
        case (2 x v'')
        then obtain v'
          where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
            and a20:"(∀i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                | Some (MValue val) ⇒ (case t of MTArray n MTypes ⇒ False | MTValue typ ⇒ MCon t v'
(hash destl (ShowL_nat i)))
                | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 | MTValue Types
⇒ False))"
            and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
        then have a40:"(∀i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v' (hash destl
(ShowL_nat i)) | _ ⇒ False)
                          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 |
MTValue Types ⇒ False))"
          using 2 by fastforce
        have a55:"(∃p. accessStore destl v' = Some (MPointer p)) ∨ accessStore destl v' = None"
          using mdef minitSingleChange_typedPref[OF a10] unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def by fastforce

        show ?case
        proof intros
          fix i
          assume iless:"i<Suc x"

          show "case accessStore (hash destl (ShowL_nat i)) v'' of None ⇒ False
                | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash destl (ShowL_nat i)) | _
⇒ False)
             | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
          proof(cases "i < x")
            case True

            then have MCondestl:"MCon (MTArray x t) v' destl" using a40 MCon.simps(2)[of x t v' destl]
a55 by simp

            then show ?thesis
            proof(cases t)
              case mtr:(MTArray x11 x12)
              then have b10:"MCon t v'' (hash destl (ShowL_nat x)) ∧
                          (∀destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore
destl' v' = accessStore destl' v'')"
                using MTArray a30 minitRec_SubPrefixes agt by blast
              then have b15:"(hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)
                              ∧ ¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))
                              ⟶ accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'')" by simp
              have " (ShowL_nat i) ≠ (ShowL_nat x)" using True
                by (metis Read_Show_nat'_id less_irrefl_nat)
              then have b16:"hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)"
                by (simp add: hash_never_equal_suffix)
              have b17:"¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))" using
ShowLNatDot hash_int_prefix b16 by simp
              have b18:"¬ LSubPrefL2 (hash destl (ShowL_nat x)) (hash destl (ShowL_nat i))" using
ShowLNatDot hash_int_prefix b16 by simp
              have b19:"∀destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore destl'
v' = accessStore destl' v''"
                using b10 by auto
              have accessSt:"accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v''"
                by (simp add: b15 b16 b17)
              then obtain val where valdef:" accessStore (hash destl (ShowL_nat i)) v' = Some (MPointer
val)"
```

```
              using MConArrayPointers[of x x11 x12 v' destl i] a40 MCondestl True mtr by auto
            have "Suc x - 1 = x" by simp
            then have subPref:"TypedMemSubPref (hash destl (ShowL_nat i)) destl (MTArray x t)"
              using MCon_array_decrement[of "Suc x" t ] True by auto
            then have b30:"( ∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l)"
              using minitSelfPointers a10 by blast
            then have valeq:"val = (hash destl (ShowL_nat i))" using True valdef subPref by simp
            then have b20:"MCon t v' (hash destl (ShowL_nat i))" using a40 mtr True valdef by
fastforce
            then have " ∀l l'. TypedMemSubPref l (hash destl (ShowL_nat i)) t ∧ accessStore l v' =
Some (MPointer l') ⟶ l' = l"
              using b30 moreSpecificTypedSubPref[of destl x t v' ] True by blast
            then have "MCon t
                v'' (hash destl (ShowL_nat i))" using MCon_mem_preserved_disjoint_update[of "(hash
destl
                    (ShowL_nat i))" "(hash destl (ShowL_nat x))" t v' v'' ] a10 b17 b18 b20 b19
              by blast
            then show ?thesis using accessSt valdef mtr valeq by (metis mtypes.simps(5)
memoryvalue.simps(6) Option.option.simps(5))
          next
            case (MTValue x2)
            then have b10:"MCon t v'' (hash destl (ShowL_nat x)) ∧
                        (∀destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore
destl' v' = accessStore destl' v'')"
              using MTArray a30 minitRec_SubPrefixes agt by blast
            then have b15:"(hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)
                            ∧ ¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))
                            ⟶ accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'')" by simp
            have " (ShowL_nat i) ≠ (ShowL_nat x)" using True
              by (metis Read_Show_nat'_id less_irrefl_nat)
            then have b16:"hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)"
              by (simp add: hash_never_equal_suffix)

            have b17:"¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))" using
ShowLNatDot hash_int_prefix b16 by simp
            have b18:"¬ LSubPrefL2 (hash destl (ShowL_nat x)) (hash destl (ShowL_nat i))" using
ShowLNatDot hash_int_prefix b16 by simp
            have b20:"MCon t v' (hash destl (ShowL_nat i))" using a40 MTValue True by (auto
split:option.splits memoryvalue.splits mtypes.splits)
            then have "MCon t
                v'' (hash destl (ShowL_nat i))" using MCon_mem_preserved_disjoint_update[of "hash
destl
                    (ShowL_nat i)" "hash destl (ShowL_nat x)" t v' v'']  using MTValue b10 b17 b18
              by simp
            moreover have "accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'"
              by (simp add: b15 b16 b17)
            moreover obtain val where "accessStore (hash destl (ShowL_nat i)) v' = Some (MValue
val)"
              using a40 MTValue True b20 by (auto split:option.splits memoryvalue.splits
mtypes.splits)
            ultimately show ?thesis
              by (simp add: MTValue b15 b16 b17)
          qed


        next
          case False
          then have iIsx:"i = x" using iless by simp
          then show ?thesis
          proof(cases t)
            case mtr:(MTArray x11 x12)
```

```
            then have b25:"v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash
destl (ShowL_nat x))) (MTArray x11 x12) v'
                                in iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12) m
x11)"
              using a30 minitRec.simps(1)[of " (hash destl (ShowL_nat x))" x11 x12] by simp
            have mc:"MCon t v'' (hash destl (ShowL_nat x))" using MTArray[of "(hash destl (ShowL_nat
x))" v' v''] a30 agt by simp
            then obtain m' where b30:"m' = updateTypedStore (hash destl (ShowL_nat x)) (MPointer
(hash destl (ShowL_nat x))) (MTArray x11 x12) v'" by auto
            then have b40:"v'' = iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i))
x12) m' x11" using b25 by metis
            have b50:"accessStore (hash destl (ShowL_nat i)) m' = Some (MPointer (hash destl (ShowL_nat
x)))" using b30 iIsx
              unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
              by auto
            have "∀suffs. hash (hash destl (ShowL_nat x)) suffs ≠ hash destl (ShowL_nat x)" using
hash_inequality by simp
            then have b60:"accessStore (hash destl (ShowL_nat i)) v'' = accessStore (hash destl
(ShowL_nat i)) m' "
              using b40 iIsx
              by (metis minitSingleChange_typedPref)

            then show ?thesis using b50 b60 mtr mc by simp
          next
            case (MTValue x2)
            then have b10:"v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2))
(MTValue x2) v'"
              using a30 minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2] by simp
            then have b40:"accessStore (hash destl (ShowL_nat i)) v'' = Some ((MValue (ival x2)))"
              using iIsx unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by auto
            then have "typeCon x2 (ival x2)"
              using ivalTypeCon by auto
            then show ?thesis using iIsx MTValue iless MCon.simps(1) b40 by simp
          qed
        qed
      qed

      moreover have "(∃p. accessStore destl v''' = Some (MPointer p)) ∨ accessStore destl v''' =
None"

        using "**" accessStore_updateStore minitRec.simps(1) minitSingleChange_typedPref
        unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
        by (metis (lifting) store.select_convs(1) typedstore.surjective typedstore.update_convs(1)
minitRec.simps(1) updateStore_def updateTypeStore_def
            updateTypedStore_def)
      ultimately show ?thesis using MCon.simps(2)[of x1 t v''' destl] by (simp add: True)
    next
      case False
      then show ?thesis using *** by simp
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    assume ***:"minitRec destl (MTValue x) a = v''''"
      and *:"arraysGreaterZero (MTValue x)"
    have v''Def:"updateTypedStore destl (MValue (ival x)) (MTValue x) a = v''''" using *** unfolding
minitRec.simps by simp
    moreover have "accessStore destl v''' = Some (MValue (ival x))" using calculation
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def by auto
    moreover have "typeCon x (ival x)"
```

```
      by (simp add: ivalTypeCon)
    ultimately show "MCon (MTValue x) v''' destl" unfolding MCon.simps
      using v''Def by (auto split:option.splits memoryvalue.splits)
  qed
qed

lemma MConMinit:
  assumes "iter (λi m''. minitRec (hash ld (ShowL_nat i)) t m'') md x = updM"
    and "accessStore (ld) md = None"
    and "arraysGreaterZero (MTArray x t)"
  shows " MCon (MTArray x t) updM ld" using assms(2,3)
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter (λi. minitRec (hash ld (ShowL_nat i)) t) md x = v'"
      and a20:"(accessStore ld md = None ⟶ arraysGreaterZero (MTArray x t) ⟶ MCon (MTArray x t) v'
ld)"
      and a30:"minitRec (hash ld (ShowL_nat x)) t v' = v''" by blast

  have sameACC:" (∀destl'. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStore destl' v' =
accessStore destl' v'')"
    using minitRec_SubPrefixes a30 by blast
  then have "accessStore ld v' = accessStore ld v''"
    using a30
    by (metis LSubPrefL2_def hash_inequality hash_suffixes_associative)


  then have ld:"(∃p. accessStore (ld) v'' = Some (MPointer p)) ∨ accessStore (ld) v'' = None"
  proof(cases "x =0")
    case True
    then have v'MD:"v' = md" using a10 by auto
    then have "minitRec (hash ld (ShowL_nat x)) t md = v''" using a30 by blast
    then have "accessStore ld md = accessStore ld v''"
      using v'MD using <accessStore ld v' = accessStore ld v''> by blast
    then show ?thesis using assms(2) by simp
  next
    case False
    then have "arraysGreaterZero (MTArray x t)" using 2(4) by simp
    then have "MCon (MTArray x t) v' ld" using a20 2 False by blast
    then show ?thesis
      by (metis <accessStore ld v' = accessStore ld v''> MCon.simps(2))
  qed


  have "∀i<Suc x.
            case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
             | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
  proof intros
    fix i assume *:"i<Suc x"
    show "case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
          | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
    proof(cases t)
      case (MTArray x11 x12)
      then show ?thesis
      proof(cases "i<x")
```

```
      case True
      then have "arraysGreaterZero (MTArray x t)" using 2(4) by simp
      then have b10: "MCon (MTArray x t) v' ld" using a20 2
        using True by fastforce
      then obtain loc2 where l2Def:"accessStore (hash ld (ShowL_nat i)) v' = Some (MPointer loc2)"
        using True MCon.simps(2) MTArray MConArrayPointers
        by meson
      have b50:"¬ LSubPrefL2 (hash ld (ShowL_nat i)) (hash ld (ShowL_nat x))" using True
        by (metis Read_Show_nat'_id ShowLNatDot hash_int_prefix hash_never_equal_suffix nat_neq_iff)
      have b60:" ¬ LSubPrefL2 (hash ld (ShowL_nat x)) (hash ld (ShowL_nat i))" using True
        using b50 ShowLNatDot hash_int_prefix by force
      then have b65:"accessStore (hash ld (ShowL_nat i)) v' = accessStore (hash ld (ShowL_nat i)) v''"
        using minitRec_SubPrefixes a30 b50 by blast
      then have "accessStore (hash ld (ShowL_nat i)) v'' = Some (MPointer loc2)" using l2Def by simp
      have b70:" (∀ l l'. TypedMemSubPref l ld (MTArray x t) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)"
        using minitSelfPointers a10 by blast
      have "∀ i<x. case accessStore (hash ld (ShowL_nat i)) v' of None ⇒ False
            | Some (MValue val) ⇒ (case t of MTArray n MTypes ⇒ False | MTValue typ ⇒ MCon t v'
(hash ld (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 | MTValue Types
⇒ False)"
        using b10 by auto
      then have b80:"MCon t v' loc2" using b10 MCon.simps(2)[of x t v' ld] True l2Def MTArray True
        by fastforce
      have "TypedMemSubPref (hash ld (ShowL_nat i)) ld (MTArray x t)"
        using True by auto
      then have loc2Same:"loc2 = (hash ld (ShowL_nat i))" using b70 l2Def by metis
      then have "MCon t
          v'' (hash ld (ShowL_nat i))" using MCon_mem_preserved_disjoint_update[of "(hash ld
          (ShowL_nat i))" "(hash ld (ShowL_nat x))" "t" v' v''] b50 b60 b70 b80 b65
        using True a30 minitRec_SubPrefixes TypedMemSubPref.simps(2) by blast
      moreover have "accessStore (hash ld (ShowL_nat i)) v'' = Some (MPointer loc2)" using b65 l2Def
by auto
      ultimately show ?thesis using MTArray True loc2Same by (auto split:memoryvalue.splits
option.splits mtypes.splits)
    next
      case False
      then have iIsx:"i = x" using * by auto
      have "arraysGreaterZero t" using 2(4) by simp
      then have MCON:"MCon t v'' (hash ld (ShowL_nat x))"
        using minitRecMCon[of "(hash ld (ShowL_nat x))" t v' v''] a30 iIsx by simp
      have v''Def:" (let m = updateTypedStore (hash ld (ShowL_nat x)) (MPointer (hash ld (ShowL_nat
x))) (MTArray x11 x12) v'
                      in iter (λi. minitRec (hash (hash ld (ShowL_nat x)) (ShowL_nat i)) x12) m x11)
                        = v''"
        using a30 minitRec.simps(1)[of "(hash ld (ShowL_nat x))" x11 x12] iIsx
        by (simp add: MTArray)
      then obtain m where mDef:"m = updateTypedStore (hash ld (ShowL_nat x)) (MPointer (hash ld
(ShowL_nat x))) (MTArray x11 x12) v'" by simp
      then have v''Def:"v'' = iter (λi. minitRec (hash (hash ld (ShowL_nat x)) (ShowL_nat i)) x12) m
x11"
        using v''Def by presburger
      have "(∀ suffix. hash (hash ld (ShowL_nat x)) suffix ≠ hash ld (ShowL_nat x))" using
hash_inequality by simp
      then have "accessStore (hash ld (ShowL_nat x)) m = accessStore (hash ld (ShowL_nat x)) v''"
        using minitSingleChange_typedPref v''Def by blast
      then have "accessStore (hash ld (ShowL_nat i)) v'' = Some (MPointer (hash ld (ShowL_nat i)))"
        using iIsx mDef unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
      then show ?thesis using MCON MTArray iIsx by (auto split:memoryvalue.splits option.splits
mtypes.splits)
    qed
  next
```

```
      case (MTValue x2)
      then show ?thesis
      proof(cases "i<x")
        case True
        then have "arraysGreaterZero (MTArray x t)" using 2(4) by simp
        then have "MCon (MTArray x t) v' ld" using a20 2
          using True by blast
        then have MCON:"case accessStore (hash ld (ShowL_nat i)) v' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v' (hash ld (ShowL_nat i)) | _ ⇒
False)
             | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 | MTValue Types
⇒ False)"
          using True MCon.simps(2)[of "x" t v' ld]
          by simp
        then obtain v where vdef:"accessStore (hash ld (ShowL_nat i)) v' = Some (MValue v)"
          using True MTValue by (auto split:option.splits memoryvalue.splits mtypes.splits)
        then have tc:"typeCon x2 v" using MCon.simps vdef True MTValue MCON by auto
        have b10:"(ShowL_nat i) ≠ (ShowL_nat x)" using True
          by (simp add: Read_Show_nat'_id readLintNotEqual)
        then have b20:"(∀x'. (ShowL_nat i) ≠ hash (ShowL_nat x) x')" using True
          by (metis LSubPrefL2_def hash_inequality ShowLNatDot hash_int_prefix
hash_suffixes_associative)
        then have "accessStore (hash ld (ShowL_nat i)) v' = accessStore (hash ld (ShowL_nat i)) v''"
          using a30 b10 MTValue accessStore_non_changed hash_never_equal_suffix minitRec.simps(2)
          using ShowLNatDot hash_int_prefix sameACC by presburger
        then have v''Store:"accessStore (hash ld (ShowL_nat i)) v'' = Some (MValue v)"
          using vdef by simp
        then show ?thesis using MCon.simps vdef True MTValue v''Store tc by auto
      next
        case False

        then have iIsx: "i = x" using * by simp
        have sv'':"v'' = updateTypedStore (hash ld (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v'"
          using minitRec.simps(2)[of "(hash ld (ShowL_nat x))" x2] a30 MTValue by simp
        then have v''Store:"accessStore (hash ld (ShowL_nat i)) v'' = Some (MValue (ival x2))"
          using sv'' iIsx unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def
          by (auto split:option.splits memoryvalue.splits mtypes.splits)
        have "typeCon x2 ((ival x2))"
          by (simp add: ivalTypeCon)
        then show ?thesis using MCon.simps iIsx MTValue v''Store by auto
      qed
    qed
  qed
  then show ?case using MCon.simps(2)[of "Suc x" t v'' "(hash ld (ShowL_nat x))"] ld by simp
qed
```

9.4 Existing locations and the top location

```
lemma minitRecSomeOldNew:
  shows "minitRec destl t destm = v'
         ⟶ (∀l. (TypedMemSubPref l destl t ∨ l = destl) ⟶ accessStore l destm = None)
         ⟶ (∀l accx. accessStore l destm = Some accx ⟶ accessStore l v' = Some accx)"
proof(induction t arbitrary: destl destm v')
  case (MTArray x1 t)
  show ?case
  proof intros
    fix l accx
    assume *:"minitRec destl (MTArray x1 t) destm = v'"
      and ***:"∀l. TypedMemSubPref l destl (MTArray x1 t) ∨ l = destl ⟶ accessStore l destm = None"
      and **:"accessStore l destm = Some accx "
    have a10:"v' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm in iter (λi.
minitRec (hash destl (ShowL_nat i)) t) m x1)"
      using * minitRec.simps(1)[of destl x1 t ] by simp
```

```
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm" by
auto
    then have a45:"accessStore destl m = Some(MPointer destl)" unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def by simp
    then have a50:"v' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a10 a40 by
presburger
    show "accessStore l v' = Some accx" using a45 ** ***
    proof(induction rule: iter_ind[OF _ _ a50[symmetric]])
      case (1)
      then show ?case
        by (metis "***" a40 minitRec.simps(1) minitRec_SubPrefixes_typedPref minitSingleChange_typedPref
option.distinct(1))
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(accessStore destl m = Some (MPointer destl) ⟶
          accessStore l destm = Some accx ⟶
          (∀ l l'. TypedMemSubPref l destl (MTArray x t) ∨ l = destl ⟶ accessStore l destm = None)
⟶ accessStore l v' = Some accx)"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by metis
      then have "accessStore destl m = Some (MPointer destl)" using 2 by blast
      then show ?case
      proof(cases "l = destl")
        case True
        then show ?thesis using a20 2
          by fastforce
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref l destl (MTArray x t)")
          case True
          have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
          then have b50:"accessStore l v' = accessStore l v''"
            using * a30 by (meson minitRec_SubPrefixes)
          then show ?thesis using a20 2 using True
            by (metis less_Suc_eq TypedMemSubPref.simps(2))
        next
          case False
          have b10:"accessStore l destm = Some accx" using 2(4) by auto

          then show ?thesis
            by (metis "2.IH" "2.prems"(3) TypedMemSubPref.simps(2) a45 less_Suc_eq
minitRec_SubPrefixes_typedPref option.distinct(1))
        qed
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix l accx
    assume *:"minitRec destl (MTValue x) destm = v'"
      and ***:"∀ l. TypedMemSubPref l destl (MTValue x) ∨ l = destl ⟶ accessStore l destm = None"
      and **:" accessStore l destm = Some accx"
    have "v' = updateTypedStore destl (MValue (ival x)) (MTValue x) destm"
      using minitRec.simps(2)[of destl x ] * by simp
    then obtain v where b10: "accessStore destl v' = Some (MValue v)"
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
    show "accessStore l v' = Some accx"
      by (metis "*" "**" "***" is_none_code(2) is_none_simps(1) minitRec_SubPrefixes_typedPref)
```

```
  qed
qed


lemma minitSomeOldNew:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
    and "∀l. (TypedMemSubPref l destl (MTArray x1 t) ∨ l = destl) ⟶ accessStore l m = None"
  shows "(∀l accx. accessStore l m = Some accx ⟶ accessStore l v' = Some accx)" using assms(2)
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
      and a20: "((∀l. TypedMemSubPref l destl (MTArray x t) ∨ l = destl ⟶ accessStore l m = None)
⟶
          (∀l accx. accessStore l m = Some accx ⟶ accessStore l v' = Some accx))"
      and a30: "minitRec (hash destl (ShowL_nat x)) t v' = v''" by metis
  show ?case
  proof intros
    fix l accx
    assume *:"accessStore l m = Some accx "
    show "accessStore l v'' = Some accx"
    proof(cases t)
      case (MTArray x11 x12)

      then show ?thesis
      proof(cases "TypedMemSubPref l destl (MTArray x t)")
        case True
        have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
        then have b50:"accessStore l v' = accessStore l v''"
          using * minitRec_SubPrefixes a30 by blast
        then show ?thesis using a20 True * b50
          using a10
          by (metis "2.prems" less_Suc_eq TypedMemSubPref.simps(2))
      next
        case False
        have "∀l. TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
            (ShowL_nat x) ⟶ ¬ TypedMemSubPref l destl (MTArray x t)"
          using TypedMemSubPref_index_disjoint typedPrefix_imp_SubPref LSubPrefL2_refl by blast
        then have cc1:"∀l . TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
            (ShowL_nat x) ⟶ accessStore l m = accessStore l v'"
          using minitSingleChange_typedPref[OF a10] by blast
        then have cc2:"(∀l. TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
            (ShowL_nat x) ⟶ accessStore l v' = None)"
          using 2(3) MTArray
          by (metis lessI TypedMemSubPref.simps(2))
        have cc3:"(∀l accx. accessStore l v' = Some accx ⟶ accessStore l v'' = Some accx)"
          using a30 minitRecSomeOldNew[of "(hash destl (ShowL_nat x))" t v' v''] cc2 by blast
        then have "accessStore l v' = Some accx" using * cc1
          by (metis False a10 minitSingleChange_typedPref)
        then have "accessStore l v'' = Some accx" using cc3 by simp
        then show ?thesis by blast
      qed
    next
      case (MTValue x2)
      then have "v'' = updateTypedStore (hash destl (ShowL_nat x)) (MValue (ival x2)) (MTValue x2) v'"
        using a30 minitRec.simps(2)[of "(hash destl (ShowL_nat x))" x2 ] by simp
      then show ?thesis using *
        by (metis "2.hyps" "2.prems" minitSingleChange_typedPref option.distinct(1))
    qed
  qed
```

```
qed

lemma minitRectoploc:
  shows "minitRec destl t destm = v'
         ⟶ Toploc destm = Toploc v'"
proof(induction t arbitrary: destl destm v')
  case (MTArray x1 t)
  show ?case
  proof intros
    assume *:"minitRec destl (MTArray x1 t) destm = v'"
    have a10:"v' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm in iter (λi.
minitRec (hash destl (ShowL_nat i)) t) m x1)"
      using * minitRec.simps(1)[of destl x1 t ] by simp
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm" by
auto
    then have a45:"accessStore destl m = Some(MPointer destl)" unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def by simp
    have a46:"Toploc destm = Toploc m" using a40 unfolding updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by simp
    then have a50:"v' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1" using a10 a40 by
presburger
    show "Toploc destm = Toploc v'" using a46
    proof(induction rule: iter_ind[OF _ _ a50[symmetric]])
      case (1)
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(Toploc destm = Toploc m ⟶ Toploc destm = Toploc v')"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
      then have "Toploc destm = Toploc v'" using 2 by blast
      then show ?case
        by (metis "2.IH" MTArray a46)
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix l l' st
    assume *:"minitRec destl (MTValue x) destm = v'"

    have "v' = updateTypedStore destl (MValue (ival x)) (MTValue x) destm"
      using minitRec.simps(2)[of destl x ] * by simp
    then show "Toploc destm = Toploc v'" unfolding updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by simp
  qed
qed

lemma minitToploc:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
  shows "Toploc m = Toploc v'"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by fastforce
next
  case (2 x v'')
  then obtain v'
    where a10: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
      and a20: " Toploc m = Toploc v'"
      and a30: "minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
  then show ?case using minitRectoploc by metis
qed
```

9.5 Types of initialised locations

```
lemma minit_TypeCompChangeIndexs:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
  shows "∀ i<x1. accessTypeStore (hash destl (ShowL_nat i)) v' = Some t"
proof (rule iter_indexed_type_writes[
    where loc = "λi. hash destl (ShowL_nat i)" and tp = t, OF assms])
  fix i j v1 v2
  assume j_less: "j < i"
    and step: "minitRec (hash destl (ShowL_nat i)) t v1 = v2"
  show "accessTypeStore (hash destl (ShowL_nat j)) v2 =
      accessTypeStore (hash destl (ShowL_nat j)) v1"
    using j_less step
    by (metis ShowLNatDot hash_int_prefix hashesIntSame minitRec_SubPrefixes_typed nat_neq_iff)
next
  fix i v1 v2
  assume step: "minitRec (hash destl (ShowL_nat i)) t v1 = v2"
  show "accessTypeStore (hash destl (ShowL_nat i)) v2 = Some t"
  proof (cases t)
    case (MTArray x11 x12)
    have step_arr:
      "minitRec (hash destl (ShowL_nat i)) (MTArray x11 x12) v1 = v2"
      using step MTArray by simp
    from minitRec_array_root_both[OF step_arr]
    show ?thesis
      using MTArray by force
  next
    case (MTValue x2)
    then show ?thesis
      using step minitRec.simps(2) by force
  qed
qed


lemma minitRec_TypeCompChange:
  "minitRec destl tp a = v''' ⟶
  (∀ destl'.
TypedMemSubPref destl' destl tp ⟶
  (case tp of MTValue val ⇒ accessTypeStore destl v''' = Some tp
      | MTArray x' t' ⇒ (∃ t''. CompMemType v''' x' t' t'' destl destl' ∧
                          (case t'' of MTArray parent_len parent_arr ⇒
                              ∀ i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v''' = Some
parent_arr
                           | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
                       )))"
proof(induction tp arbitrary:destl a v''' )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:"minitRec destl (MTArray x1 t) a = v'''"
      and ***:"TypedMemSubPref destl' destl (MTArray x1 t)"
    have a5:"(let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a
                    in iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1) = v'''"
      using ** minitRec.simps(1)[of destl x1 t ] by simp
    then obtain m where mdef:"updateTypedStore destl (MPointer destl) (MTArray x1 t) a = m" by auto

    then have v''def:"v''' = iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x1"
      using a5 mdef by metis
    have "∃ t''. CompMemType v''' x1 t t'' destl destl' ∧
              (case t'' of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
               | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))"
      using ***
```

```
    proof(induction rule: iter_ind[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
          and a20:"(TypedMemSubPref destl' destl (MTArray x t) ⟶
            (∃t''. CompMemType v' x t t'' destl destl' ∧
                  (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v' = Some parent_arr
                   | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
          and a30:"minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast
      then show ?case
      proof(cases "destl' = destl")
        case True
        then show ?thesis using 2
          by (metis accessStore_updateStore hash_inequality minitSelfPointers
minitSingleChange_typedPref)
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
          case True
          then have "destl' ≠ hash destl (ShowL_nat x)"
            by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
          moreover have "∀i<x. ¬ TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
t" using True
            using TypedMemSubPref_hash_index_disjoint_typed by blast
          ultimately have i1:"∀i<x. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore
(hash destl' (ShowL_nat i)) v''"
            using minitRec_SubPrefixes_typedPref_typed[of "(hash destl (ShowL_nat x))" t v' v''] a30
            by (meson ShowLNatDot f1 hash_injective)
          obtain t'' where i2:"(CompMemType v' x t t'' destl destl' ∧
             (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))"
            using True a20 by blast

          have i5:"∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
            using a30 minitRec_SubPrefixes by blast
          moreover have i6:"∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l (hash destl
(ShowL_nat x))"
            using TypedMemSubPref_index_disjoint by blast
          ultimately have i7:"∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =
accessStore l v''" by auto
          have i8:"∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer
l') ⟶ l' = l"
            using minitSelfPointers[OF a10] by blast

          show ?thesis
          proof(cases t'')
            case (MTArray x11 x12)
            then have cc0:"CompMemType v' x t (MTArray x11 x12) destl destl' ∧ (∀i<x11.
accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
              using i2 by simp
            then have "CompMemType v'' x t (MTArray x11 x12) destl destl' ∧ (∀i<x11. accessTypeStore
(hash destl' (ShowL_nat i)) v'' = Some x12)"
              using CompMemType_preservation_induction[OF i5 i6 i7 cc0 i8]
              using minitRec_SubPrefixes_typed a30 TypedMemSubPref_hash_not_below_index[OF True] by
blast
            then show ?thesis using CompMemType_extend_array by force
```

```
          next
            case (MTValue x2)
            then have cc0:"accessTypeStore destl' v' = Some (MTValue x2) ∧ CompMemType v' x t (MTValue
x2) destl destl'" using i2 by simp
            then have "CompMemType v'' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' =
Some (MTValue x2)"
              using CompMemTypeValue_preservation_induction[OF i5 i6 i7 cc0 i8]
              using True a30 i6 minitRec_SubPrefixes_typed by blast
            then show ?thesis using CompMemType_extend by fastforce
          qed

        next
          case False
          then have b5:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl
(ShowL_nat x)"
            using 2 TypedMemSubPref.simps(2)[of destl' destl "(Suc x)" t] f1
            using less_Suc_eq
            by (metis TypedMemSubPref.simps(2))
          then have "accessTypeStore destl' m = accessTypeStore destl' v'"
            using False a10 minitSingleChange_typedPref_typed by blast
          then have "minitRec (hash destl (ShowL_nat x)) t v' = v''" using a30 by simp
          then have IH1:"(TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
          (case t of
           MTArray x' t' ⇒
             ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                   (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                    | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
         | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t))"
            using MTArray.IH[of "(hash destl (ShowL_nat x))" v' v''] by simp

          have "∃t''. CompMemType v'' (Suc x) t t'' destl destl' ∧
            (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
          proof(cases "t")
            case (MTArray x11 x12)
            have step_arr:
              "minitRec (hash destl (ShowL_nat x)) (MTArray x11 x12) v' = v''"
              using a30 MTArray by simp
            have v''Def:
              "v'' = iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
                (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
                  (MTArray x11 x12) v') x11"
              using step_arr unfolding Let_def minitRec.simps(1) by blast
            have root_bundle:
              "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
                ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
              using minitRec_array_root_both[OF step_arr] by simp
            then show ?thesis
            proof(cases "destl' = hash destl (ShowL_nat x)")
              case True
              then have "CompMemType v'' (Suc x) t t destl destl'"
                using MTArray root_bundle by auto
              moreover have "∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12"
                using minit_TypeCompChangeIndexs[OF v''Def[symmetric]] True by auto
              ultimately show ?thesis using MTArray by force
            next
              case False
              then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using b5 by auto
              then have "∃t''. CompMemType v'' x11 x12 t'' (hash destl (ShowL_nat x)) destl' ∧
               (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
```

```
                | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
                using IH1 MTArray by simp
            then show ?thesis using MTArray root_bundle by auto
          qed
        next
          case (MTValue x2)
          then show ?thesis
            using IH1 b5 by auto
        qed

        then show ?thesis by blast
      qed
    qed
  qed
  then show "case MTArray x1 t of
     MTArray x' t' ⇒
       ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
             (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
     | MTValue val ⇒ accessTypeStore destl v''' = Some (MTArray x1 t)" by auto
qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) a = v''''"
    assume ***:"TypedMemSubPref destl' destl (MTValue x)"
    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) a"
      using ** minitRec.simps(2)[of destl x] by simp
    then have "accessTypeStore destl v''' = Some (MTValue x)"
      unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
    then show "case MTValue x of
       MTArray x' t' ⇒
         ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
               (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
                | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
       | MTValue val ⇒ accessTypeStore destl v''' = Some (MTValue x)" by simp
  qed
qed

lemma minit_TypeCompChange:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
  shows "(∀destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶
          (∃st. CompMemType v' x1 t st destl destl' ∧
            (case st of MTArray parent_len parent_arr ⇒
              ∀i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
      and a20: "(∀destl'.
             TypedMemSubPref destl' destl (MTArray x t) ⟶
             (∃st. CompMemType v' x t st destl destl' ∧
                   (case st of
                    MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v' = Some parent_arr
```

```
                  | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
      and a30: "minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast

  show ?case
  proof intros
    fix destl'
    assume a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"
    show "∃st. CompMemType v'' (Suc x) t st destl destl' ∧
           (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
    proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
      case True
      then have "destl' ≠ hash destl (ShowL_nat x)" using True
        by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
      moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using True
        using TypedMemSubPref_index_disjoint_typed by blast
      ultimately have i1:"∀destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl'
          (hash destl (ShowL_nat x)) t ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v''"
        using minitRec_SubPrefixes_typedPref_typed[of "(hash destl (ShowL_nat x))" t v' v''] a30 by simp
      obtain st where i2:"(CompMemType v' x t st destl destl' ∧
            (case st of MTArray parent_len parent_arr ⇒
               ∀i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))" using True a20 by
blast

      have self_pointers: "∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
        using minitSelfPointers[OF a10] by blast


      have store_preserved: "∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs
v' = accessStore locs v''"
        using a30 minitRec_SubPrefixes by blast
      moreover have no_prefix_conflict: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l
(hash destl (ShowL_nat x))"
        using TypedMemSubPref_index_disjoint by blast
      ultimately have store_consistency: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l
v' = accessStore l v''" by auto
      show ?thesis
      proof(cases st)
        case (MTArray x11 x12)
        have i1':"∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore (hash destl'
(ShowL_nat i)) v''"
          using i1
          using TypedMemSubPref_hash_not_below_index[OF True] TypedMemSubPref_hash_index_disjoint_typed[OF
True] LSubPrefL2_refl by metis
        then have "CompMemType v' x t (MTArray x11 x12) destl destl' ∧
                   (∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
          using i2 MTArray by simp
        then have "CompMemType v'' x t (MTArray x11 x12) destl destl'
                  ∧ (∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12)"
          using CompMemType_preservation_induction[OF store_preserved no_prefix_conflict
              store_consistency _self_pointers i1']
          by blast
        then show ?thesis using CompMemType_extend_array MTArray by force
      next
        case (MTValue x2)
        then have "CompMemType v' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v' = Some
(MTValue x2)"
          using i2 by auto
        then have "CompMemType v'' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' = Some
```

```
(MTValue x2)"
          using CompMemTypeValue_preservation_induction[OF store_preserved no_prefix_conflict
              store_consistency _self_pointers] using i1
          using ‹¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t› ‹destl' ≠ hash destl (ShowL_nat
x)› by blast
        then show ?thesis using CompMemType_extend by fastforce
      qed
    next
      case False
      then have a2:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl (ShowL_nat
x)"
        using a1 unfolding TypedMemSubPref.simps
        using less_SucE by blast
      then show ?thesis
      proof(cases t)
        case (MTArray x11 x12)
        have step_arr:
          "minitRec (hash destl (ShowL_nat x)) (MTArray x11 x12) v' = v''"
          using a30 MTArray by simp
        have v''Def:
          "v'' = iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
            (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
              (MTArray x11 x12) v') x11"
          using step_arr unfolding Let_def minitRec.simps(1) by blast
        have root_bundle:
          "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
            ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat x)))"
          using minitRec_array_root_both[OF step_arr] by simp
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
          case True
          then have "∀i<x11. TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
(MTArray x11 x12)" by auto
          then have "∀i<x11. accessTypeStore (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) v'' = Some
x12"
            using minit_TypeCompChangeIndexs[OF v''Def[symmetric]] by blast
          moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
            using MTArray True root_bundle unfolding CompMemType.simps by blast
          ultimately show ?thesis
            using MTArray True root_bundle by force
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using a2 by blast
          then have "(case t of
         MTArray x' t' ⇒
           ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                 (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                     | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
          | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t)"
            using minitRec_TypeCompChange[of "(hash destl (ShowL_nat x))" t v' v''] a30 by simp
          then show ?thesis using MTArray root_bundle by auto
        qed
      next
        case (MTValue x2)
        then have "destl' = hash destl (ShowL_nat x)" using a2 by simp
        then have "updateTypedStore destl' (MValue (ival x2)) (MTValue x2) v' = v''"
          using a30 minitRec.simps(2)[of destl' x2] MTValue by simp
        then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
          using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def by auto
      qed
    qed
  qed
```

qed

lemma *minitRec_somesome:*
"minitRec destl tp a = v''' ⟶
(∀ destl'. TypedMemSubPref destl' destl tp ⟶ ((∃ t. accessStore destl' v''' = Some t)
⟷ (∃ tt. accessTypeStore destl' v''' = Some tt)))"
proof(*induction tp arbitrary:destl a v''' *)
case *(MTArray x1 t)*
show *?case*
proof *intros*
fix *destl'*
assume ***:"minitRec destl (MTArray x1 t) a = v''''"*
and ****:"TypedMemSubPref destl' destl (MTArray x1 t)"*
have *a5:"(let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a*
in iter (λi. minitRec (hash destl (ShowL$_{nat}$ i)) t) m x1) = v''''"
using *** minitRec.simps(1)[of destl x1 t]* **by** *simp*
then obtain *m* **where** *mdef:"updateTypedStore destl (MPointer destl) (MTArray x1 t) a = m"* **by** *auto*

then have *v''def:"v''' = iter (λi. minitRec (hash destl (ShowL$_{nat}$ i)) t) m x1"*
using *a5 mdef* **by** *metis*
have *"(∃ t. accessStore destl' v''' = Some t) = (∃ tt. accessTypeStore destl' v''' = Some tt)"*
using *****
proof(*induction rule: iter_ind[OF _ _ v''def[symmetric]]*)
case *(1)*
then show *?case* **by** *simp*
next
case *(2 x v'')*
then obtain *v'*
where *a10:"iter (λi. minitRec (hash destl (ShowL$_{nat}$ i)) t) m x = v'"*
and *a20:"(TypedMemSubPref destl' destl (MTArray x t) ⟶*
(∃ t. accessStore destl' v' = Some t) = (∃ tt. accessTypeStore destl' v' = Some tt))"
and *a30:"minitRec (hash destl (ShowL$_{nat}$ x)) t v' = v''"* **by** *blast*
then show *?case*
proof(*cases "destl' = destl"*)
case *True*
then show *?thesis* **using** *2*
by *(metis accessStore_updateStore hash_inequality minitSelfPointers*
minitSingleChange_typedPref)
next
case *f1:False*
then show *?thesis*
proof(*cases "TypedMemSubPref destl' destl (MTArray x t)"*)
case *True*
then have *"destl' ≠ hash destl (ShowL$_{nat}$ x)"*
by *(metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame*
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
moreover have *"∀ i<x. ¬ TypedMemSubPref (hash destl' (ShowL$_{nat}$ i)) (hash destl (ShowL$_{nat}$ x))*
t" **using** *True*
using *TypedMemSubPref_hash_index_disjoint_typed* **by** *blast*
ultimately have *i1:"∀ i<x. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v' = accessTypeStore*
(hash destl' (ShowL$_{nat}$ i)) v''"
using *minitRec_SubPrefixes_typedPref_typed[of "(hash destl (ShowL$_{nat}$ x))" t v' v''] a30*
by *(meson ShowLNatDot f1 hash_injective)*
have *i2:"(∃ t. accessStore destl' v' = Some t) = (∃ tt. accessTypeStore destl' v' = Some tt)"*
using *True a20* **by** *blast*

have *i5:"∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL$_{nat}$ x)) ⟶ accessStore locs v' =*
accessStore locs v''"
using *a30 minitRec_SubPrefixes* **by** *blast*
moreover have *i6:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l (hash destl*
(ShowL$_{nat}$ x))"
using *TypedMemSubPref_index_disjoint* **by** *blast*
ultimately have *i7:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =*
accessStore l v''" **by** *auto*

```
        have i8:"∀ l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer
l') ⟶ l' = l"
          using minitSelfPointers[OF a10] by blast

        show ?thesis using i2
          using True a30 i6 i7 minitRec_SubPrefixes_typed by auto
      next
        case False
        then have b5:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl
(ShowL_nat x)"
          using 2 TypedMemSubPref.simps(2)[of destl' destl "(Suc x)" t] f1
          using less_Suc_eq
          by (metis TypedMemSubPref.simps(2))
        then have "accessTypeStore destl' m = accessTypeStore destl' v'"
          using False a10 minitSingleChange_typedPref_typed by blast
        then have "minitRec (hash destl (ShowL_nat x)) t v' = v''" using a30 by simp
        then have IH1:"(∀ destl'.
      TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
      (∃ t. accessStore destl' v'' = Some t) = (∃ tt. accessTypeStore destl' v'' = Some tt))"
          using MTArray.IH[of "(hash destl (ShowL_nat x))" v' v''] by simp

        have "(∃ t. accessStore destl' v'' = Some t) = (∃ tt. accessTypeStore destl' v'' = Some tt)"
        proof(cases "t")
          case (MTArray x11 x12)
          have step_arr:
            "minitRec (hash destl (ShowL_nat x)) (MTArray x11 x12) v' = v''"
            using a30 MTArray by simp
          have v''Def:
            "v'' = iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
              (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
                (MTArray x11 x12) v') x11"
            using step_arr unfolding Let_def minitRec.simps(1) by blast
          have root_bundle:
            "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
              ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
            using minitRec_array_root_both[OF step_arr] by simp
          then show ?thesis
          proof(cases "destl' = hash destl (ShowL_nat x)")
            case True
            then have "CompMemType v'' (Suc x) t t destl destl'"
              using MTArray root_bundle by auto
            moreover have "∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12"
              using minit_TypeCompChangeIndexs[OF v''Def[symmetric]] True by auto
            ultimately show ?thesis
              using MTArray True root_bundle by auto
          next
            case False
            then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using b5 by auto
            then have "(∃ t. accessStore destl' v'' = Some t) = (∃ tt. accessTypeStore destl' v'' =
Some tt)"
              using IH1 MTArray by simp
            then show ?thesis using MTArray root_bundle by auto
          qed
        next
          case (MTValue x2)
          then show ?thesis
            using IH1 b5 by auto
        qed

        then show ?thesis by blast
      qed
    qed
  qed
```

```
    then show "(∃t. accessStore destl' v''' = Some t) = (∃tt. accessTypeStore destl' v''' = Some tt)"
by auto
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:" minitRec destl (MTValue x) a = v''''"
    assume ***:"TypedMemSubPref destl' destl (MTValue x)"
    then have mdef:"v''' = updateTypedStore destl (MValue (ival x)) (MTValue x) a"
      using ** minitRec.simps(2)[of destl x] by simp
    then have "accessTypeStore destl v''' = Some (MTValue x)"
      unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
    then show "(∃t. accessStore destl' v''' = Some t) = (∃tt. accessTypeStore destl' v''' = Some tt)"
      by (metis "***" MConAccessSame.simps(1) SameMCon_imps_MConAccessSame TypedMemSubPref.simps(1)
arraysGreaterZero.simps(1) mdef
          minitRec.simps(2) minitRecMCon)
  qed
qed

lemma minit_TypeCompChange_somesome:
  assumes "iter (λi m''. minitRec (hash destl (ShowL_nat i)) t m'') m x1 = v'"
  shows "(∀destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶
          ((∃t. accessStore destl' v' = Some t) ⟷ (∃tt. accessTypeStore destl' v' = Some tt)))"
proof(induction rule: iter_ind[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter (λi. minitRec (hash destl (ShowL_nat i)) t) m x = v'"
      and a20: "(∀destl'. TypedMemSubPref destl' destl (MTArray x t) ⟶
            ((∃t. accessStore destl' v' = Some t) ⟷ (∃tt. accessTypeStore destl' v' = Some tt)))"
      and a30: "minitRec (hash destl (ShowL_nat x)) t v' = v''" by blast

  show ?case
  proof intros
    fix destl'
    assume a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"
    show "((∃t. accessStore destl' v'' = Some t) ⟷ (∃tt. accessTypeStore destl' v'' = Some tt))"
    proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
      case True
      then have "destl' ≠ hash destl (ShowL_nat x)" using True
        by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
      moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using True
        using TypedMemSubPref_index_disjoint_typed by blast
      ultimately have i1:"∀destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl'
          (hash destl (ShowL_nat x)) t ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v''"
        using minitRec_SubPrefixes_typedPref_typed[of "(hash destl (ShowL_nat x))" t v' v''] a30 by simp
      have i2:"(∃t. accessStore destl' v' = Some t) = (∃tt. accessTypeStore destl' v' = Some tt)"
        using True a20 by blast

      have store_preserved: "∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs
v' = accessStore locs v''"
        using a30 minitRec_SubPrefixes by blast
      moreover have no_prefix_conflict: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l
(hash destl (ShowL_nat x))"
        using TypedMemSubPref_index_disjoint by blast
      ultimately have store_consistency: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l
v' = accessStore l v''"
```

```
        by auto

      show ?thesis using i2
        using True ‹¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t› ‹destl' ≠ hash
            destl (ShowL_nat x)› i1 store_consistency by force

    next
      case False
      then have a2:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl (ShowL_nat
x)"
        using a1 unfolding TypedMemSubPref.simps
        using less_SucE by blast
      then show ?thesis
      proof(cases t)
        case (MTArray x11 x12)
        have step_arr:
          "minitRec (hash destl (ShowL_nat x)) (MTArray x11 x12) v' = v''"
          using a30 MTArray by simp
        have v''Def:
          "v'' = iter (λi. minitRec (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12)
            (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
              (MTArray x11 x12) v') x11"
          using step_arr unfolding Let_def minitRec.simps(1) by blast
        have root_bundle:
          "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
            ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat x)))"
          using minitRec_array_root_both[OF step_arr] by simp
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
          case True
          then have "∀i<x11. TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
(MTArray x11 x12)" by auto
          then have "∀i<x11. accessTypeStore (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) v'' = Some
x12"
            using minit_TypeCompChangeIndexs[OF v''Def[symmetric]] by blast
          moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
            using MTArray True root_bundle unfolding CompMemType.simps by blast
          ultimately show ?thesis
            using MTArray True root_bundle by simp
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using a2 by blast
          then have "(∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' = Some
tt)"
            using minitRec_somesome[of "(hash destl (ShowL_nat x))" t v' v''] a30 by simp
          then show ?thesis using MTArray root_bundle by auto
        qed
      next
        case (MTValue x2)
        then have "destl' = hash destl (ShowL_nat x)" using a2 by simp
        then have "updateTypedStore destl' (MValue (ival x2)) (MTValue x2) v' = v''"
          using a30 minitRec.simps(2)[of destl' x2] MTValue by simp
        then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
          using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def
          by (metis TypedMemSubPref.simps(1) a30 minitRec_somesome)
      qed
    qed
  qed
qed


end
```

10 Compatibility of storage and memory types (TypeSafe_Mem_Sto_Comp)

theory *TypeSafe_Mem_Sto_Comp*
imports *TypeSafe_Memory TypeSafe_Storage*
begin

Relates *cps2mTypeCompatible* to *cps2mTypeConvert* and transfers typed sub-prefixes between storage and memory types.

lemma *convertComp:*
assumes *"cps2mTypeCompatible t' tp"*
shows *"cps2mTypeConvert t' = Some tp"* **using** *assms*
proof(*induction t' arbitrary:tp*)
case *(STArray x1 t')*
then obtain *x12* **where** *tpDef:"tp = MTArray x1 x12 ∧ cps2mTypeCompatible t' x12"*
using *cps2mTypeCompatible.simps*
by *(metis extractType.cases)*
then show *?case* **using** *STArray.IH* **by** *auto*
next
case *(STMap x1 t')*
then show *?case* **by** *simp*
next
case *(STValue x)*
then show *?case*
by *(metis mtypes.exhaust cps2mTypeCompatible.simps(1,6) cps2mTypeConvert.simps(1))*
qed

lemma *compatible_TypedStoSubpref_imps_TypedMemSubPref:*
assumes *"cps2mTypeCompatible st mt"*
shows *"TypedStoSubpref x' l st ⟷ TypedMemSubPref x' l mt ∨ x' = l"*
proof
assume *"TypedStoSubpref x' l st"*
then show *"TypedMemSubPref x' l mt ∨ x' = l"* **using** *assms*
proof(*induction st arbitrary:mt l*)
case *(STArray x1 st)*
then show *?case*
by *(metis stypes.inject(1) TypedMemSubPref.simps(2) TypedStoSubpref.simps(2)*
cps2mTypeCompatible.elims(2) cps2mTypeCompatible.simps(3))
next
case *(STMap x1 st)*
then show *?case* **by** *simp*
next
case *(STValue x)*
then show *?case*
by *(metis TypedStoSubpref.simps(1))*
qed
next
assume *"TypedMemSubPref x' l mt ∨ x' = l"*
then show *"TypedStoSubpref x' l st"* **using** *assms*
proof(*cases "x' = l"*)
case *True*
then show *"TypedStoSubpref x' l st"* **using** *TypedStoSubpref_sameLoc* **by** *simp*
next
case *False*
then have *a1:"TypedMemSubPref x' l mt"*
using ‹*TypedMemSubPref x' l mt ∨ x' = l*› **by** *blast*
then show *"TypedStoSubpref x' l st"* **using** *assms*
proof(*induction mt arbitrary: st l*)
case *(MTArray x1 mt)*
then obtain *st'* **where** *st_def: "st = STArray x1 st'"* **and** *compat: "cps2mTypeCompatible st' mt"*
using *cps2mTypeCompatible.elims(2)* **by** *blast*
then obtain *i* **where** *idef: "i<x1 ∧ (TypedMemSubPref x' (hash l (ShowL$_{nat}$ i)) mt ∨ x' = hash l*

```
(ShowL_nat i))"
        unfolding TypedMemSubPref.simps
        using MTArray.prems(1) by fastforce
      then show ?case
      proof(cases "x' = hash l (ShowL_nat i)")
        case True
        then show ?thesis using st_def
          using TypedStoSubpref_sameLoc idef by auto
      next
        case False
        then have sub: "TypedMemSubPref x' (hash l (ShowL_nat i)) mt" using idef by blast
        then have "TypedStoSubpref x' (hash l (ShowL_nat i)) st'" using MTArray.IH compat by blast
        then show ?thesis using idef st_def
          by auto
      qed
    next
      case (MTValue x)
      then show ?case
        by (metis stypes.exhaust TypedMemSubPref.simps(1) TypedStoSubpref.simps(1)
cps2mTypeCompatible.simps(3,4))
    qed
  qed
qed

lemma compatible_TypedStoSubpref_imps_TypedMemSubPref_neg:
  assumes "cps2mTypeCompatible st mt"
  shows "¬ TypedStoSubpref x' l st ⟷ ¬ TypedMemSubPref x' l mt ∧ x' ≠ l"
  using assms compatible_TypedStoSubpref_imps_TypedMemSubPref by auto

end
```

11 Copies between memory and storage (TypeSafe_Memory_Copies)

```
theory TypeSafe_Memory_Copies
  imports TypeSafe_Support TypeSafe_Mem_Sto_Comp
begin
```

11.1 Consistency Preservation Under Disjoint Locations

If two locations *destl* and *destl'* are disjoint – neither is a sub-prefix of the other under *LSubPrefL2* – then an update to *destl* cannot affect *MCon* at *destl'*. This lemma is the foundation for all sub-prefix preservation arguments: it shows that the *MCon* invariant is inherited at unrelated locations.

```
lemma MCon_preserved_under_disjoint_location:
  assumes "(¬LSubPrefL2 destl' destl) ∧ ¬LSubPrefL2 destl destl' "
  shows "(∀l l'. TypedStoSubpref l destl' t ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l)
         ∧ cps2mTypeCompatible t t'
         ∧ (∀destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl' v'')
         ∧ MCon t' v' destl' ⟶ MCon t' v'' destl'" using assms
proof (induction t' arbitrary:destl' t)
  case (MTArray x1 t')
  show ?case
  proof intros
    assume " (∀l l'. TypedStoSubpref l destl' t ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l) ∧
    cps2mTypeCompatible t (MTArray x1 t') ∧ (∀destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl'
v' =
        accessStore destl' v'') ∧ MCon (MTArray x1 t') v' destl'"
    then have a10:"(∀l l'. TypedStoSubpref l destl' t ∧ accessStore l v' = Some (MPointer l') ⟶ l' =
l)"
      and a20:"cps2mTypeCompatible t (MTArray x1 t')"
      and a30:"(∀destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore destl'
v'')"
      and a40:"MCon (MTArray x1 t') v' destl'" by blast+
    show "MCon (MTArray x1 t') v'' destl'"
```

```
    proof(cases "x1>0")
      case True
      have "∀ i<x1.
             case accessStore (hash destl' (ShowL_nat i)) v'' of None ⇒ False
              | Some (MValue val) ⇒ (case t' of MTValue typ ⇒ MCon t' v'' (hash destl' (ShowL_nat i)) |
_ ⇒ False)
              | Some (MPointer loc2) ⇒ (case t' of MTArray len' arr' ⇒ MCon t' v'' loc2 | MTValue
Types ⇒ False)"
      proof (intros)
        fix i assume iLess:"i<x1"
        then show " case accessStore (hash destl' (ShowL_nat i)) v'' of None ⇒ False
              | Some (MValue val) ⇒ (case t' of MTValue typ ⇒ MCon t' v'' (hash destl' (ShowL_nat i)) |
_ ⇒ False)
              | Some (MPointer loc2) ⇒ (case t' of MTArray len' arr' ⇒ MCon t' v'' loc2 | MTValue
Types ⇒ False)"
        proof(cases t')
          case mtr:(MTArray x11 x12)
          then obtain t''' where tdef:"t = STArray x1 (STArray x11 t''')" using a20 mtr
cps2mTypeCompatible.simps True
            by (metis stypes.exhaust)
          then have a45:"cps2mTypeCompatible (STArray x11 t''') (MTArray x11 x12)" using a20 by (simp
add: mtr)

          have b100:"¬ LSubPrefL2 (hash destl' (ShowL_nat i)) destl" using Mutual_NonSub_SpecificNonSub
MTArray by auto
          then have "¬ LSubPrefL2 destl (hash destl' (ShowL_nat i))" using Not_Sub_More_Specific
MTArray by auto
          then have a50:"(∀ l l'. TypedStoSubpref l (hash destl' (ShowL_nat i)) (STArray x11 t''') ∧
              accessStore l v' = Some (MPointer l') ⟶ l' = l) ∧
                    cps2mTypeCompatible (STArray x11 t''') (MTArray x11 x12) ∧ (∀ destl'. ¬ LSubPrefL2
destl' destl ⟶ accessStore destl' v' = accessStore destl' v'')
                    ∧ MCon (MTArray x11 x12) v' (hash destl' (ShowL_nat i)) ⟶
                    MCon (MTArray x11 x12) v'' (hash destl' (ShowL_nat i))"
            using MTArray(1)[of "(hash destl' (ShowL_nat i))" "(STArray x11 t''')"] b100 mtr by blast
          have a55:"∀ l l'. TypedStoSubpref l (hash destl' (ShowL_nat i)) (STArray x11 t''') ∧
accessStore l v' = Some (MPointer l') ⟶ l' = l"
            using a10 stoMoreSpecificTypedSubpref[of destl' x1 "(STArray x11 t''')" v'] iLess using
tdef by blast
          then have a60:"accessStore (hash destl' (ShowL_nat i)) v' = accessStore (hash destl' (ShowL_nat
i)) v''" using b100 a30 by blast
          then show ?thesis
          proof(cases "accessStore (hash destl' (ShowL_nat i)) v'")
            case None
            then show ?thesis using a10 iLess a40 by auto
          next
            case (Some a)
            then show ?thesis
            proof(cases a)
              case (MValue x1)
              then show ?thesis using a40 iLess mtr Some by auto
            next
              case (MPointer x2')
              then have x2'def:"x2' = (hash destl' (ShowL_nat i))" using a55 Some
                using iLess TypedStoSubpref.simps(2) a10 tdef by blast
              have "∀ i<x1.
               case accessStore (hash destl' (ShowL_nat i)) v' of None ⇒ False
               | Some (MValue val) ⇒ (case t' of MTArray n MTypes ⇒ False | MTValue typ ⇒ MCon t' v'
(hash destl' (ShowL_nat i)))
               | Some (MPointer loc2) ⇒ (case t' of MTArray len' arr' ⇒ MCon t' v' loc2 | MTValue Types
⇒ False)"
                using MCon.simps(2)[of x1 t' v' destl']
                using a40 by auto
              then have "MCon t' v' (x2')" using a40 iLess Some mtr MPointer tdef MCon.simps(2)[of x1
t' v' destl'] True by fastforce
```

```
            then have "MCon t' v'' x2'" using a50 b100 a55 a30 a45 tdef x2'def mtr by blast
            then show ?thesis using Some MPointer mtr a60 by auto
          qed
        qed
      next
        case (MTValue x2)
        then have tdef:"t = STArray x1 (STValue x2)" using a20 cps2mTypeCompatible.simps True
          by (metis stypes.exhaust)
        then have a45:"cps2mTypeCompatible (STValue x2) (MTValue x2)" using a20 by (simp add:
MTValue)
        have b100:"¬ LSubPrefL2 (hash destl' (ShowL_nat i)) destl" using Mutual_NonSub_SpecificNonSub
MTArray by auto
        then have "¬ LSubPrefL2 destl (hash destl' (ShowL_nat i))" using Not_Sub_More_Specific
MTArray by auto
        then have a50:"(∀ l l'. TypedStoSubpref l (hash destl' (ShowL_nat i)) (STValue x2)
                ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l)
                ∧ cps2mTypeCompatible (STValue x2) (MTValue x2)
                ∧ (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v' = accessStore
destl' v'')
                  ∧ MCon (MTValue x2) v' (hash destl' (ShowL_nat i)) ⟶
                  MCon (MTValue x2) v'' (hash destl' (ShowL_nat i))
          " using MTArray(1)[of "(hash destl' (ShowL_nat i))"] b100 using MTValue by force
        then have a60:"accessStore (hash destl' (ShowL_nat i)) v' = accessStore (hash destl' (ShowL_nat
i)) v''" using b100 a30 by simp
        then show ?thesis
        proof(cases "accessStore (hash destl' (ShowL_nat i)) v'")
          case None
          then show ?thesis using iLess a40 by auto
        next
          case (Some a)
          then show ?thesis
          proof(cases a)
            case (MValue x1)
            then have "MCon t' v' (hash destl' (ShowL_nat i))" using a40 iLess Some MTValue by auto
            then have "MCon t' v'' (hash destl' (ShowL_nat i))" using a50 b100 a60 a30 Some MValue
MTValue by auto
            then show ?thesis using Some MValue MTValue a60 by auto
          next
            case (MPointer x2')
            then show ?thesis using a40 iLess MTValue Some by auto
          qed

        qed
      qed
    qed
    moreover have "(∃p. accessStore destl' v'' = Some (MPointer p)) ∨ accessStore destl' v'' =
None"
      using MTArray.prems True a10 a20 a30 a40 by force
    ultimately show ?thesis using MCon.simps(2)[of x1 t' v'' destl']
      by (simp add: True)
  next
    case False
    then show ?thesis using a40 by simp
  qed
qed
next
  case (MTValue x)
  show ?case
  proof intros
    assume *:"(∀ l l'. TypedStoSubpref l destl' t ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l)
∧
    cps2mTypeCompatible t (MTValue x) ∧ (∀ destl'. ¬ LSubPrefL2 destl' destl ⟶ accessStore destl' v'
= accessStore destl' v'') ∧ MCon (MTValue x) v' destl'"
    then have "accessStore destl' v' = accessStore destl' v''" using MTValue by simp
```

```
    then show "MCon (MTValue x) v'' destl'" using * by simp
  qed
qed
```

11.2 Destination Root Preservation

The top-level entry at the destination root location *destl* is unchanged after any recursive copy step. Concretely, *accessStore destl* and *accessTypeStore destl* return the same value before and after the call.

cpm2m works by iterating *iter'* over each array index: at each step it calls *cpm2mrec* on one index, and *cpm2mrec* may recursively call *iter'* for nested arrays. To support the mutual induction the following root-preservation lemmas are established first, covering all three copy directions: *cpm2mrec* (memory to memory), *cps2mrec* (storage to memory), and *cpm2srec* (memory to storage).

updateTypedStore only writes to the single location *destl*, so any *destl'* with *destl' ≠ destl* is unchanged in both value and type stores.

```
lemma updateTypedStore_neq_loc:
  "destl' ≠ destl
   ⟹ accessStore destl' (updateTypedStore destl v t m) = accessStore destl' m
     ∧ accessTypeStore destl' (updateTypedStore destl v t m) = accessTypeStore destl' m"
  unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def
  by auto

lemma cpm2mrec_preserves_dest_root_both:
  shows "(cpm2mrec (hash ls suffixa) (hash ld suffix') t ms m = Some v'')
    ⟶ accessStore ld v'' = accessStore ld m
      ∧ accessTypeStore ld v'' = accessTypeStore ld m"
proof (induction t arbitrary: m v'' suffixa suffix' ls)
  case (MTArray x1 t)
  show ?case
  proof
    assume run:"cpm2mrec (hash ls suffixa) (hash ld suffix') (MTArray x1 t) ms m = Some v''"
    then obtain l where src_ptr:"accessStore (hash ls suffixa) ms = Some (MPointer l)"
      using cpm2mrec.simps(1) by (auto split: if_splits option.splits memoryvalue.splits)
    have expanded:"Some v'' =
       (let m0 = updateTypedStore (hash ld suffix') (MPointer (hash ld suffix')) (MTArray x1 t) m
        in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms) m0
x1)"
      using run src_ptr
      by (simp add: cpm2mrec.simps(1) split: if_splits option.splits memoryvalue.splits)
    then obtain m0 where m0_def:
      "m0 = updateTypedStore (hash ld suffix') (MPointer (hash ld suffix')) (MTArray x1 t) m" by simp
    have init:
      "accessStore ld m0 = accessStore ld m ∧ accessTypeStore ld m0 = accessTypeStore ld m"
      using m0_def hash_inequality updateTypedStore_neq_loc
      by metis
    have iter_run:
      "iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms) m0 x1 =
Some v''"
      using expanded m0_def
      by presburger
    have step:
      "⋀i m1 m2.
        accessStore ld m1 = accessStore ld m ∧ accessTypeStore ld m1 = accessTypeStore ld m
        ⟹ cpm2mrec (hash l (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms m1 = Some m2
        ⟹ accessStore ld m2 = accessStore ld m ∧ accessTypeStore ld m2 = accessTypeStore ld m"
    proof -
      fix i m1 m2
      assume inv:
        "accessStore ld m1 = accessStore ld m ∧ accessTypeStore ld m1 = accessTypeStore ld m"
      assume rec:
        "cpm2mrec (hash l (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms m1 = Some m2"
      have rec':
```

```
        "cpm2mrec (hash l (ShowL_nat i)) (hash ld (hash suffix' (ShowL_nat i))) t ms m1 = Some m2"
        using hash_suffixes_associative rec by simp
      then have
        "accessStore ld m2 = accessStore ld m1 ∧ accessTypeStore ld m2 = accessTypeStore ld m1"
        using MTArray.IH by blast
      with inv show
        "accessStore ld m2 = accessStore ld m ∧ accessTypeStore ld m2 = accessTypeStore ld m"
        by auto
    qed
    show "accessStore ld v'' = accessStore ld m ∧ accessTypeStore ld v'' = accessTypeStore ld m"
      using iter'_invariant[
        where I = "λm1. accessStore ld m1 = accessStore ld m ∧ accessTypeStore ld m1 = accessTypeStore
ld m",
        OF iter_run init step]
      by simp
  qed
next
  case (MTValue x)
  show ?case
  proof
    assume run:"cpm2mrec (hash ls suffixa) (hash ld suffix') (MTValue x) ms m = Some v''"
    then obtain v where src_val:"accessStore (hash ls suffixa) ms = Some (MValue v)"
      using cpm2mrec.simps(2) by (auto split: option.splits memoryvalue.splits)
    have "v'' = updateTypedStore (hash ld suffix') (MValue v) (MTValue x) m"
      using run src_val cpm2mrec.simps(2)[of "hash ls suffixa" "hash ld suffix'" x ms m] by simp
    then show "accessStore ld v'' = accessStore ld m ∧ accessTypeStore ld v'' = accessTypeStore ld m"
      using hash_inequality updateTypedStore_neq_loc by metis
  qed
qed

lemma cpm2mrec_preserves_dest_root:
  shows "(cpm2mrec (hash ls suffixa) (hash ld suffix') t ms m = Some v'')⟶ accessStore ld v'' =
accessStore ld m"
  using cpm2mrec_preserves_dest_root_both by blast

lemma cpm2mrec_preserves_dest_root_typed:
  shows "(cpm2mrec (hash ls suffixa) (hash ld suffix') t ms m = Some v'')⟶ accessTypeStore ld v'' =
accessTypeStore ld m"
  using cpm2mrec_preserves_dest_root_both by blast

lemma cps2mrec_preserves_dest_root_both:
  shows "(cps2mrec (hash ls suffixa) (hash ld suffix') t ms m' = Some v'')
    ⟶ accessStore ld v'' = accessStore ld m'
      ∧ accessTypeStore ld v'' = accessTypeStore ld m'"
proof (induction t arbitrary: m' v'' suffixa suffix' ls)
  case (STArray x1 t)
  show ?case
  proof
    assume run: "cps2mrec (hash ls suffixa) (hash ld suffix') (STArray x1 t) ms m' = Some v''"
    have expanded: "Some v'' =
      (let m0 = updateTypedStore (hash ld suffix') (MPointer (hash ld suffix'))
         (MTArray x1 (case cps2mTypeConvert t of Some t' ⇒ t' | None ⇒ MTValue TBool)) m'
       in iter' (λi. cps2mrec (hash (hash ls suffixa) (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat
i)) t ms) m0 x1)"
      using run by (simp add: cps2mrec.simps(1) split: option.splits memoryvalue.splits)
    then obtain m0 where m0_def:
      "m0 = updateTypedStore (hash ld suffix') (MPointer (hash ld suffix'))
        (MTArray x1 (case cps2mTypeConvert t of Some t' ⇒ t' | None ⇒ MTValue TBool)) m'" by simp
    have init:
      "accessStore ld m0 = accessStore ld m' ∧ accessTypeStore ld m0 = accessTypeStore ld m'"
      using m0_def hash_inequality updateTypedStore_neq_loc by metis
    have iter_run:
      "iter' (λi. cps2mrec (hash (hash ls suffixa) (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t
ms) m0 x1 = Some v''"
```

```
      using expanded m0_def
      by presburger
    have step:
      "⋀i m1 m2.
        accessStore ld m1 = accessStore ld m' ∧ accessTypeStore ld m1 = accessTypeStore ld m'
        ⟹ cps2mrec (hash (hash ls suffixa) (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms m1
= Some m2
        ⟹ accessStore ld m2 = accessStore ld m' ∧ accessTypeStore ld m2 = accessTypeStore ld m'"
    proof -
      fix i m1 m2
      assume inv:
        "accessStore ld m1 = accessStore ld m' ∧ accessTypeStore ld m1 = accessTypeStore ld m'"
      assume rec:
        "cps2mrec (hash (hash ls suffixa) (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t ms m1 =
Some m2"
      have rec':
        "cps2mrec (hash ls (hash suffixa (ShowL_nat i))) (hash ld (hash suffix' (ShowL_nat i))) t ms m1 =
Some m2"
        using hash_suffixes_associative rec by simp
      then have
        "accessStore ld m2 = accessStore ld m1 ∧ accessTypeStore ld m2 = accessTypeStore ld m1"
        using STArray.IH by blast
      with inv show
        "accessStore ld m2 = accessStore ld m' ∧ accessTypeStore ld m2 = accessTypeStore ld m'"
        by auto
    qed
    show "accessStore ld v'' = accessStore ld m' ∧ accessTypeStore ld v'' = accessTypeStore ld m'"
      using iter'_invariant[
        where I = "λm1. accessStore ld m1 = accessStore ld m' ∧ accessTypeStore ld m1 =
accessTypeStore ld m'",
        OF iter_run init step]
      by simp
  qed
next
  case (STValue x)
  show ?case
  proof
    assume run: "cps2mrec (hash ls suffixa) (hash ld suffix') (STValue x) ms m' = Some v''"
    then obtain v where src_val: "accessStorage x (hash ls suffixa) ms = v"
      using cps2mrec.simps(2)[of "hash ls suffixa" "hash ld suffix'" x ms m']
      by (auto split: option.splits memoryvalue.splits)
    have "v'' = updateTypedStore (hash ld suffix') (MValue v) (MTValue x) m'"
      using run src_val cps2mrec.simps(2)[of "hash ls suffixa" "hash ld suffix'" x ms m'] by simp
    then show "accessStore ld v'' = accessStore ld m' ∧ accessTypeStore ld v'' = accessTypeStore ld
m'"
      using hash_inequality updateTypedStore_neq_loc by metis
  qed
next
  case (STMap k v)
  then show ?case using cps2mrec.simps(3) by auto
qed

lemma cps2mrec_preserves_dest_root:
  shows "(cps2mrec (hash ls suffixa) (hash ld suffix') t ms m' = Some v'')⟶ accessStore ld v'' =
accessStore ld m'"
  using cps2mrec_preserves_dest_root_both by blast

lemma cpm2srec_preserves_dest_root:
  shows "∀t'. (cpm2srec (hash ls suffixa) (hash ld suffix') t ms m' = Some v'')⟶ accessStorage t' ld
v'' = accessStorage t' ld m'"
proof intros
  fix t'
  assume **:"cpm2srec (hash ls suffixa) (hash ld suffix') t ms m' = Some v''"
  then show "accessStorage t' ld v'' = accessStorage t' ld m'"
```

```
  proof (induction t arbitrary: m' v'' suffixa suffix' ls)
    case (MTArray x1 t)
    then obtain l where ldef:"accessStore (hash ls suffixa) ms = Some (MPointer l)" unfolding
cpm2srec.simps by (simp split:option.splits memoryvalue.splits)
    then have a60:"Some v'' = iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash (hash ld suffix')
(ShowL_nat i)) t ms) m' x1"
      using MTArray unfolding cpm2srec.simps(1) by (simp split:option.splits memoryvalue.splits)
    show ?case
    proof(induction rule: iter'_induct[OF _ _ a60[symmetric]])
      case (1 v')
      then have "v' = m'" using iter'.simps by simp
      then show ?case by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash (hash ld suffix') (ShowL_nat i)) t
ms) m' x = Some v'"
          and a20:"accessStorage t' ld v' = accessStorage t' ld m'"
          and a30:"cpm2srec (hash l (ShowL_nat x)) (hash (hash ld suffix') (ShowL_nat x)) t ms v' = Some
v''" by blast
      then show ?case
        using MTArray.IH hash_suffixes_associative by auto
    qed
  next
    case (MTValue x)
    then have "hash ld suffix' ≠ ld"
      by (simp add: hash_inequality)
    then show ?case using MTValue unfolding cpm2srec.simps accessStorage_def by (auto
split:option.splits memoryvalue.splits)
  qed
qed
```

11.3 Single-Step Change for Recursive Copy Functions

Each call to *cpm2mrec* or *cps2mrec* (a single recursive copy step) only modifies locations that are sub-prefixes of the destination index being written. Any location whose suffix differs from the destination suffix is left unchanged.

These lemmas underpin the iterated single-change results: by showing the invariant holds at every recursive step, it can be propagated across the full *iter'* loop. The *iterIndexMinOne* helper establishes that a successful iteration of length *x* implies the existence of an intermediate result after *x* steps.

```
lemma iterIndexMinOne:
  assumes "iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md (Suc x) = Some
v''"
  shows "∃v'. iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md x = Some v'"
proof -
  have "Some v'' =(if Suc x ≤ 0 then Some md
     else case iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md (Suc x - 1) of
None ⇒ None
          | Some xa ⇒ cpm2mrec (hash ls (ShowL_nat (Suc x - 1))) (hash ld (ShowL_nat (Suc x - 1))) t ms
xa)"
    using assms iter'.simps[of " (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms)" md
"(Suc x)"] by simp
  then have **:"Some v'' = (case iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t
ms) md (Suc x - 1) of None ⇒ None
          | Some xa ⇒ cpm2mrec (hash ls (ShowL_nat (Suc x - 1))) (hash ld (ShowL_nat (Suc x - 1))) t ms
xa)" by simp
  then show ?thesis
  proof(cases "iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md (Suc x - 1)")
    case None
    then show ?thesis using ** by simp
  next
    case (Some a)
    then have "Some v'' = cpm2mrec (hash ls (ShowL_nat (Suc x - 1))) (hash ld (ShowL_nat (Suc x - 1))) t
ms a" using ** by simp
```

```
    then have "iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md x =
              iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md (Suc x - 1)"
by simp
    then show ?thesis using Some by simp
  qed
qed

lemma cpm2m_iter_dest_unchanged:
  shows " Some v'' = iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem)
m x1
         ∧ (∀suffix. hash destl suffix ≠ destl) ⟶ accessStore destl m = accessStore destl v'' "
proof intros
  assume " Some v'' = iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat
      i)) t' srcMem) m x1 ∧ (∀suffix. hash destl suffix ≠ destl)"
  then have *:"Some v'' = iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t'
srcMem) m x1"
    and **: "(∀suffix. hash destl suffix ≠ destl)" by simp+
  then show "accessStore destl m = accessStore destl v''"
  proof(induction rule: iter'_induct[OF _ _ *[symmetric]])
    case (1 v')
    then show ?case by simp
  next
    case (2 x v'')
    then obtain v'
      where a10:"iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem) m x
= Some v'"
       and a20:"(Some v' = iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl
           (ShowL_nat i)) t' srcMem) m x ∧ (∀suffix. hash destl suffix ≠ destl) ⟶
           accessStore destl m = accessStore destl v')"
       and a30:"cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' srcMem v' = Some v''"
by blast
    then show ?case using cpm2mrec_preserves_dest_root[of srcl "(ShowL_nat x)" destl "(ShowL_nat x)" t'
srcMem v' v'']
      using "**" by auto
  qed
qed

lemma cps2mAccessPrePost:
  shows " Some v'' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem)
m x1
         ∧ (∀suffix. hash destl suffix ≠ destl) ⟶ accessStore destl m = accessStore destl v'' "
proof intros
  assume " Some v'' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat
      i)) t' srcMem) m x1 ∧ (∀suffix. hash destl suffix ≠ destl)"
  then have *:"Some v'' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t'
srcMem) m x1"
    and **: "(∀suffix. hash destl suffix ≠ destl)" by simp+
  then show "accessStore destl m = accessStore destl v''"
  proof(induction rule: iter'_induct[OF _ _ *[symmetric]])
    case (1 v')
    then show ?case by simp
  next
    case (2 x v'')
    then obtain v'
      where a10:"iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem) m x
= Some v'"
       and a20:"(Some v' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl
           (ShowL_nat i)) t' srcMem) m x ∧ (∀suffix. hash destl suffix ≠ destl) ⟶
           accessStore destl m = accessStore destl v')"
       and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' srcMem v' = Some v''"
by blast
    then show ?case using cps2mrec_preserves_dest_root
      using "**" by simp
  qed
```

qed

11.4 Sub-Prefix Preservation: Location-Prefix Condition

A copy operation only modifies locations that are sub-prefixes of the destination root. Any location *destl'* that is not a sub-prefix of *destl* (under *LSubPrefL2*) is left unchanged in both the value and type stores.

This section covers the *both*-access (value and type together), value-only, and typed-only variants for all three recursive copy functions: *cpm2mrec*, *cps2mrec*, and *cpm2srec*. The *cpm2srec_SubPrefixes_typedPref* lemma additionally handles the case where the type compatibility predicate (*cps2mTypeCompatible*) is involved.

The following meta-lemma lifts a pointwise step-preservation property to a full *iter'* run. It abstracts the repeated pattern in every *SubPrefixes_both* proof: the *Not_Sub_More_Specific* chain and invariant bookkeeping are done once here.

```
lemma iter'_SubPref_both:
  fixes destl :: String.literal
  assumes run: "iter' f init_m x = Some v'"
  assumes init: "∀loc. ¬LSubPrefL2 loc destl
    ⟶ accessStore loc a = accessStore loc init_m
      ∧ accessTypeStore loc a = accessTypeStore loc init_m"
  assumes step_pres: "⋀i m m'. f i m = Some m'
    ⟹ ∀loc. ¬LSubPrefL2 loc (hash destl (ShowL_nat i))
        ⟶ accessStore loc m = accessStore loc m'
          ∧ accessTypeStore loc m = accessTypeStore loc m'"
  shows "∀loc. ¬LSubPrefL2 loc destl
    ⟶ accessStore loc a = accessStore loc v'
      ∧ accessTypeStore loc a = accessTypeStore loc v'"
proof (rule iter'_invariant[OF run,
    where I = "λm1. ∀loc. ¬LSubPrefL2 loc destl
      ⟶ accessStore loc a = accessStore loc m1
        ∧ accessTypeStore loc a = accessTypeStore loc m1"])
  show "∀loc. ¬LSubPrefL2 loc destl
    ⟶ accessStore loc a = accessStore loc init_m
      ∧ accessTypeStore loc a = accessTypeStore loc init_m"
    using init .
next
  fix i m1 m2
  assume IH_iter: "∀loc. ¬LSubPrefL2 loc destl
    ⟶ accessStore loc a = accessStore loc m1
      ∧ accessTypeStore loc a = accessTypeStore loc m1"
  assume rec: "f i m1 = Some m2"
  show "∀loc. ¬LSubPrefL2 loc destl
    ⟶ accessStore loc a = accessStore loc m2
      ∧ accessTypeStore loc a = accessTypeStore loc m2"
  proof (intro allI impI)
    fix loc assume not_sub: "¬LSubPrefL2 loc destl"
    then have inv1: "accessStore loc a = accessStore loc m1 ∧ accessTypeStore loc a = accessTypeStore
loc m1"
      using IH_iter by blast
    have not_sub_child: "¬LSubPrefL2 loc (hash destl (ShowL_nat i))"
      using not_sub Not_Sub_More_Specific by simp
    then have inv2: "accessStore loc m1 = accessStore loc m2 ∧ accessTypeStore loc m1 =
accessTypeStore loc m2"
      using step_pres[OF rec] by blast
    show "accessStore loc a = accessStore loc m2 ∧ accessTypeStore loc a = accessTypeStore loc m2"
      using inv1 inv2 by simp
  qed
qed

lemma cpm2mrec_SubPrefixes_both:
  "cpm2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. (¬LSubPrefL2 destl' destl) ⟶
      accessStore destl' a = accessStore destl' v'''
    ∧ accessTypeStore destl' a = accessTypeStore destl' v''')"
```

```
proof (induction tp arbitrary: srcl destl srcMem a v''')
  case (MTArray x1 t)
  show ?case
  proof
    assume run: "cpm2mrec srcl destl (MTArray x1 t) srcMem a = Some v'''"
    then obtain l where src_ptr: "accessStore srcl srcMem = Some (MPointer l)"
      using cpm2mrec.simps(1)[of srcl destl x1 t srcMem a]
      by (auto split: if_splits option.splits memoryvalue.splits)
    have expanded:
      "Some v''' =
        (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a
         in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using run src_ptr cpm2mrec.simps(1)[of srcl destl x1 t srcMem a]
      by simp
    then obtain m where mdef: "m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto
    have init: "∀destl'. ¬LSubPrefL2 destl' destl
        ⟶ accessStore destl' a = accessStore destl' m
          ∧ accessTypeStore destl' a = accessTypeStore destl' m"
      using mdef updateTypedStore_neq_loc
      by (metis LSubPrefL2_def)
    have iter_run:
      "iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1 = Some v'''"
      using expanded mdef by presburger
    have step_pres: "⋀i m1 m2.
        cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2
        ⟹ ∀loc. ¬LSubPrefL2 loc (hash destl (ShowL_nat i))
            ⟶ accessStore loc m1 = accessStore loc m2
              ∧ accessTypeStore loc m1 = accessTypeStore loc m2"
      using MTArray.IH by blast
    show "∀destl'. ¬LSubPrefL2 destl' destl
      ⟶ accessStore destl' a = accessStore destl' v''' ∧ accessTypeStore destl' a = accessTypeStore
destl' v'''"
      using iter'_SubPref_both[OF iter_run init step_pres] .
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume run: "cpm2mrec srcl destl (MTValue x) srcMem a = Some v'''"
    assume not_sub: "¬LSubPrefL2 destl' destl"
    then have not_eq: "destl' ≠ destl" by (simp add: LSubPrefL2_def)
    then obtain v where src_val: "accessStore srcl srcMem = Some (MValue v)"
      using run cpm2mrec.simps(2)[of srcl destl x srcMem a]
      by (auto split: option.splits memoryvalue.splits)
    have as2:"v''' = updateTypedStore destl (MValue v) (MTValue x) a"
      using run src_val cpm2mrec.simps(2)[of srcl destl x srcMem a] by simp
    then show "accessStore destl' a = accessStore destl' v'''"
      using not_eq updateTypedStore_neq_loc by metis
    show "accessTypeStore destl' a = accessTypeStore destl' v'''"
      using as2 not_eq updateTypedStore_neq_loc by metis
  qed
qed

lemma cpm2mrec_SubPrefixes:
  "cpm2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. (¬LSubPrefL2 destl' destl) ⟶ accessStore destl' a = accessStore destl' v''')"
  using cpm2mrec_SubPrefixes_both by blast

lemma cps2mrec_SubPrefixes_both:
  "cps2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. (¬LSubPrefL2 destl' destl) ⟶
      accessStore destl' a = accessStore destl' v'''
    ∧ accessTypeStore destl' a = accessTypeStore destl' v''')"
```

```
proof (induction tp arbitrary: srcl destl srcMem a v''')
  case (STArray x1 t)
  show ?case
  proof
    assume run: "cps2mrec srcl destl (STArray x1 t) srcMem a = Some v'''"
    have expanded:
      "Some v''' =
        (case cps2mTypeConvert t of
          None ⇒ None
        | Some t' ⇒
            let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a
            in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using run cps2mrec.simps(1)[of srcl destl x1 t srcMem a]
      by (auto split: option.splits memoryvalue.splits)
    then obtain t' where t'Def: "cps2mTypeConvert t = Some t'" by fastforce
    then obtain m where mdef: "m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a" by auto
    have init: "∀destl'. ¬LSubPrefL2 destl' destl
        ⟶ accessStore destl' a = accessStore destl' m
          ∧ accessTypeStore destl' a = accessTypeStore destl' m"
      using mdef updateTypedStore_neq_loc by (metis LSubPrefL2_def)
    have iter_run0:
      "iter' (λi m'. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m')
        (updateTypedStore destl (MPointer destl) (MTArray x1 t') a) x1 = Some v'''"
      using run t'Def unfolding cps2mrec.simps(1) Let_def by (auto split:option.splits)
    have iter_run:
      "iter' (λi m'. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m') m x1 =
Some v'''"
      using iter_run0 mdef by simp
    have step_pres: "⋀i m1 m2.
        cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2
        ⟹ ∀loc. ¬LSubPrefL2 loc (hash destl (ShowL_nat i))
            ⟶ accessStore loc m1 = accessStore loc m2
              ∧ accessTypeStore loc m1 = accessTypeStore loc m2"
      using STArray.IH by blast
    show "∀destl'. ¬LSubPrefL2 destl' destl
      ⟶ accessStore destl' a = accessStore destl' v''' ∧ accessTypeStore destl' a = accessTypeStore
destl' v'''"
      using iter'_SubPref_both[OF iter_run init step_pres] .
  qed
next
  case (STValue x)
  show ?case
  proof intros
    fix destl'
    assume run: "cps2mrec srcl destl (STValue x) srcMem a = Some v'''"
    assume not_sub: "¬LSubPrefL2 destl' destl"
    then have not_eq: "destl' ≠ destl" by (simp add: LSubPrefL2_def)
    have result:
      "v''' = updateTypedStore destl (MValue (accessStorage x srcl srcMem)) (MTValue x) a"
      using run cps2mrec.simps(2)[of srcl destl x srcMem a] by simp
    then show "accessStore destl' a = accessStore destl' v'''"
      using not_eq updateTypedStore_neq_loc by metis
    show "accessTypeStore destl' a = accessTypeStore destl' v'''"
      using result not_eq updateTypedStore_neq_loc by metis
  qed
next
  case (STMap x1 tp)
  then show ?case using cps2mrec.simps(3) by auto
qed

lemma cps2mrec_SubPrefixes:
  "cps2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. (¬LSubPrefL2 destl' destl) ⟶ accessStore destl' a = accessStore destl' v''')"
  using cps2mrec_SubPrefixes_both by blast
```

```
lemma cpm2srec_SubPrefixes:
  "cpm2srec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl' t. (¬LSubPrefL2 destl' destl) ⟶ accessStorage t destl' a = accessStorage t destl' v''')"
proof(induction tp arbitrary:srcl destl srcMem a v''' )
  case (MTArray x1 tp)
  show ?case
  proof intros
    fix destl' t
    assume "cpm2srec srcl destl (MTArray x1 tp) srcMem a = Some v''''"
    then have
      **: "cpm2srec srcl destl (MTArray x1 tp) srcMem a = Some v''''" by auto+
    then obtain l where ldef:"accessStore srcl srcMem = Some (MPointer l)"
      using ** cpm2srec.simps(1)[of srcl destl x1 tp srcMem a ] by (auto split:if_splits option.splits
memoryvalue.splits)
    then have a5:"Some v''' = iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) tp
srcMem) a x1"
      using ** cpm2srec.simps(1)[of srcl destl x1 tp srcMem a ] by (auto split:if_splits option.splits
memoryvalue.splits)

    show "¬ LSubPrefL2 destl' destl ⟹ accessStorage t destl' a = accessStorage t destl' v''''"
    proof(induction rule: iter'_induct[OF _ _ a5[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) a x =
Some v'"
          and a20:"(¬ LSubPrefL2 destl' destl ⟶ accessStorage t destl' a = accessStorage t destl'
v')"
          and a30:"cpm2srec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) tp srcMem v' = Some v''"
by blast
      then show ?case
      proof(cases tp)
        case (MTArray x11 x12)
        then show ?thesis
          using "2.prems" MTArray.IH a20 a30 Not_Sub_More_Specific by fastforce
      next
        case (MTValue x2)
        then have a50:"cpm2srec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTValue x2) srcMem v'
= Some v''" using a30 by simp
        then obtain v where vdef: "accessStore (hash l (ShowL_nat x)) srcMem = Some (MValue v)"
          using a30 unfolding cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
        then have "Some v'' = Some (v'(hash destl (ShowL_nat x) $$:= v))" using a50 unfolding
cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
        then show ?thesis by (metis "2.prems" MTArray a20 a30 Not_Sub_More_Specific)
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix destl' t
    assume *:"cpm2srec srcl destl (MTValue x) srcMem a = Some v''''"
      and **:" ¬ LSubPrefL2 destl' destl"
    then obtain v where vdef: "accessStore srcl srcMem = Some (MValue v)"
      unfolding cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
    then have "Some v''' = Some (a(destl $$:= v))" using * unfolding cpm2srec.simps by (auto
split:if_splits option.splits memoryvalue.splits)
    moreover have "destl' ≠ destl" using ** LSubPrefL2_def by simp
    ultimately show "accessStorage t destl' a = accessStorage t destl' v''' " using **
accessStorage_def by simp
```

```
  qed
qed
```

11.5 Value Type Preservation Under Iterated Copying

When the copied type is a value type (`MTValue`/`STValue`), every destination index written by the iteration holds a value entry (not a pointer) in the resulting memory. These lemmas are used in the `MCon`/`SCon` correctness proofs to discharge the value-branch obligations.

```
lemma Cpm2mrec_val_types:
  assumes "iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) (MTValue x2) srcMem)
m x1 = Some v'"
  shows "∀i<x1. (∃val. accessStore (hash destl (ShowL_nat i)) v' = Some (MValue val))"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) (MTValue x2)
srcMem) m x = Some v'"
      and a20: "(∀i<x. ∃val. accessStore (hash destl (ShowL_nat i)) v' = Some (MValue val))"
      and a30:"cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTValue x2) srcMem v' =
Some v''" by blast
  obtain t where a40: "accessStore (hash srcl (ShowL_nat x)) srcMem = Some (MValue t)" using a30 using
cpm2mrec.simps(2) by (auto split:option.splits memoryvalue.splits)
  show ?case
  proof intros
    fix i assume *:"i<Suc x"
    show "∃val. accessStore (hash destl (ShowL_nat i)) v'' = Some (MValue val) "
    proof(cases "(i<x)")
      case True
      then have "¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))"
        by (simp add: Read_Show_nat'_id hash_int_prefix hash_never_equal_suffix readLintNotEqual
ShowLNatDot)
      then have "accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl (ShowL_nat i)) v''"
using cpm2mrec_SubPrefixes a30 a40 by force
      then show ?thesis using a20 True by metis
    next
      case False
      then have "i = x" using * by simp
      moreover have "Some v'' = (case accessStore (hash srcl (ShowL_nat x)) srcMem of None ⇒ None
                | Some (MValue v) ⇒ Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v)
(MTValue x2) v')
                | Some (MPointer literal) ⇒ None)"
        using cpm2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 srcMem v']
a30 by simp
      ultimately show ?thesis using a40
        unfolding updateTypedStore_def updateTypeStore_def accessStore_def updateStore_def by simp
    qed
  qed
qed

lemma Cps2mrec_val_types:
  assumes "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) (STValue x2) srcMem)
m x1 = Some v'"
  shows "∀i<x1. (∃val. accessStore (hash destl (ShowL_nat i)) v' = Some (MValue val))"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) (STValue x2)
```

```
srcMem) m x = Some v'"
      and a20: "(∀ i<x. ∃ val. accessStore (hash destl (ShowL_nat i)) v' = Some (MValue val))"
      and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STValue x2) srcMem v' =
Some v''" by blast
  obtain t where a40: " accessStorage x2 srcl srcMem = t" using a30 using cps2mrec.simps(2) by (auto
split:option.splits memoryvalue.splits)
  show ?case
  proof intros
    fix i assume *:"i<Suc x"
    show "∃ val. accessStore (hash destl (ShowL_nat i)) v'' = Some (MValue val) "
    proof(cases "(i<x)")
      case True
      then have "¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))"
        by (simp add: Read_Show_nat'_id hash_int_prefix hash_never_equal_suffix readLintNotEqual
ShowLNatDot)
      then have "accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl (ShowL_nat i)) v''"
using cps2mrec_SubPrefixes a30 a40 by force
      then show ?thesis using a20 True by metis
    next
      case False
      then have "i = x" using * by simp
      moreover have "Some v'' = (let v = accessStorage x2 (hash srcl (ShowL_nat x)) srcMem
     in Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2) v'))"
        using cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 srcMem v']
a30 by simp
      ultimately show ?thesis using a40 unfolding updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def
        by auto
    qed
  qed
qed
```

11.6 Pointer Self-Reference Preservation

After a copy, every pointer stored within the destination region points to itself (i.e. the pointer value equals the location at which it is stored). This self-referential invariant reflects the structural layout of arrays: sub-location pointers introduced by `cpm2mrec`/`cps2mrec` always satisfy `accessStore l v' = Some (MPointer l)`, matching the requirement imposed by `MCon`/`SCon`.

Results are given for the recursive step (`cpm2mrecPtrsPointToSelf`, `cps2mrecPtrsPointToSelf`) and lifted to the iterated wrappers (`cpm2mSelfPointers`, `cps2mSelfPointers`).

```
lemma cpm2mrecPtrsPointToSelf:
  shows "cpm2mrec srcl destl t srcMem destm = Some v'
         ⟶ (∀ l l'. (TypedMemSubPref l destl t ∨ l = destl) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)"
proof(induction t arbitrary: srcl destl srcMem destm v')
  case (MTArray x1 t)
  show ?case
  proof intros
    fix l l'
    assume *:"cpm2mrec srcl destl (MTArray x1 t) srcMem destm = Some v' "
      and **:"(TypedMemSubPref l destl (MTArray x1 t) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have a10:"Some v' = (case accessStore srcl srcMem of None ⇒ None | Some (MValue literal) ⇒ None
     | Some (MPointer l) ⇒
         let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm
         in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using * cpm2mrec.simps(1)[of srcl destl x1 t srcMem destm] by simp
    then obtain l''' where a20:"accessStore srcl srcMem = Some (MPointer l''')"
      by (auto split:option.splits memoryvalue.splits mtypes.splits)
    then have a30:"Some v' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm
        in iter' (λi. cpm2mrec (hash l''' (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using a10 by simp
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) destm" by
```

```
auto
    then have a45:"accessStore destl m = Some(MPointer destl)"
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def by simp
    then have a50:"Some v' = iter' (λi. cpm2mrec (hash l''' (ShowL_nat i)) (hash destl (ShowL_nat i)) t
srcMem) m x1"
      using a30 a40 by presburger
    show "l' = l" using a45 **
    proof(induction rule: iter'_induct[OF _ _ a50[symmetric]])
      case (1 v')
      then show ?case by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cpm2mrec (hash l''' (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x
= Some v'"
          and a20:" (accessStore destl m = Some (MPointer destl) ⟶ (TypedMemSubPref l destl (MTArray
x t) ∨ l = destl)
              ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l) "
          and a30:"cpm2mrec (hash l''' (ShowL_nat x)) (hash destl (ShowL_nat x)) t srcMem v' = Some v''"
by blast
      then have "accessStore destl m = Some (MPointer destl)" using 2 by auto
      then show ?case
      proof(cases "l = destl")
        case True
        then show ?thesis using 2(1,3,4) cpm2mrec_preserves_dest_root by auto
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref l destl (MTArray x t)")
          case True
          have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
          then have b50:"accessStore l v' = accessStore l v''"
            using * cpm2mrec_SubPrefixes a30 by simp
          then show ?thesis using a20 2 using True by auto
        next
          case False
          then have b5:"TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
              (ShowL_nat x)" using 2(4) TypedMemSubPref.simps(2)[of l destl "(Suc x)" t] f1
            using less_Suc_eq by auto
          have b10:"accessStore l v'' = Some (MPointer l')" using 2(4) by auto

          have "(∀l l'. (TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
              (ShowL_nat x)) ∧ accessStore l v'' = Some (MPointer l') ⟶ l' = l)"
            using MTArray[of "(hash l''' (ShowL_nat x))" "(hash destl (ShowL_nat x))" srcMem v' v''] a30
by simp
          then show ?thesis using b5 b10 by blast
        qed
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix l l'
    assume *:"cpm2mrec srcl destl (MTValue x) srcMem destm = Some v'"
      and **:"(TypedMemSubPref l destl (MTValue x) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have "Some v' = (case accessStore srcl srcMem of None ⇒ None
                | Some (MValue v) ⇒ Some (updateTypedStore destl (MValue v) (MTValue x) destm)
                | Some (MPointer literal) ⇒ None)"
      using cpm2mrec.simps(2)[of srcl destl x srcMem destm] * by simp
    then obtain v where b10: "accessStore destl v' = Some (MValue v)"
```

```
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
    show "l' = l" using b10 ** by auto
  qed
qed

lemma cps2mrecPtrsPointToSelf:
  shows "cps2mrec srcl destl t srcMem destm = Some v'
         ⟶ (∀l l'. (TypedStoSubpref l destl t ∨ l = destl) ∧ accessStore l v' = Some (MPointer l')
⟶ l' = l)"
proof(induction t arbitrary: srcl destl srcMem destm v')
  case (STArray x1 t)
  show ?case
  proof intros
    fix l l'
    assume *:"cps2mrec srcl destl (STArray x1 t) srcMem destm = Some v' "
      and **:"(TypedStoSubpref l destl (STArray x1 t) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have a10:"Some v' = (case cps2mTypeConvert t of None ⇒ None
     | Some t' ⇒
         let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') destm
         in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using * cps2mrec.simps(1)[of srcl destl x1 t srcMem destm] by simp
    then obtain t' where t'def:"cps2mTypeConvert t = Some t'" by fastforce
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t') destm" by
auto
    then have a45:"accessStore destl m = Some(MPointer destl)"
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by simp
    then have a50:"Some v' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t
srcMem) m x1"
      using a40 a10 t'def
      by (metis (no_types, lifting) Option.option.simps(5))
    show "l' = l" using a45 **
    proof(induction rule: iter'_induct[OF _ _ a50[symmetric]])
      case (1 v')
      then show ?case by auto
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x
= Some v'"
          and a20:" (accessStore destl m = Some (MPointer destl) ⟶ (TypedStoSubpref l destl (STArray
x t) ∨ l = destl)
              ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l) "
          and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t srcMem v' = Some v''"
by blast
      then have "accessStore destl m = Some (MPointer destl)" using 2 by auto
      then show ?case
      proof(cases "l = destl")
        case True
        then show ?thesis using 2 cpm2mrec_preserves_dest_root
          by (metis (no_types, opaque_lifting) LSubPrefL2_def cps2mrec_SubPrefixes hash_inequality
hash_suffixes_associative)
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref l destl (STArray x t)")
          case True
          then obtain i where b10:"i<x ∧ (TypedStoSubpref l (hash destl (ShowL_nat i)) t ∨ l = (hash
destl (ShowL_nat i)))"
            using f1 by auto
          then have b15:"LSubPrefL2 l (hash destl (ShowL_nat i))" using TypedStoSubpref_imp_LSubPrefL2[of
l "(hash destl
```

```
            (ShowL_nat i))" t] unfolding LSubPrefL2_def by auto
          then have b20:"(hash destl (ShowL_nat i)) ≠ (hash destl (ShowL_nat x))" using b10
            by (metis Read_Show_nat'_id hash_never_equal_suffix less_not_refl)
          have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using LSubPrefL2_def[of l "hash destl
            (ShowL_nat x)"] b15 b20 Mutual_NonSub_SpecificNonSub hash_int_prefix ShowLNatDot by auto
          then have b50:"accessStore l v' = accessStore l v''"
            using * cps2mrec_SubPrefixes a30 by simp
          then show ?thesis using a20 2 using True by auto
        next
          case False
          then have b5:"TypedStoSubpref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
            (ShowL_nat x)" using 2(4) TypedStoSubpref.simps(2)[of l destl "(Suc x)" t] f1
            using less_Suc_eq by auto
          have b10:"accessStore l v'' = Some (MPointer l')" using 2(4) by auto

          have "(∀l l'. (TypedStoSubpref l (hash destl (ShowL_nat x)) t ∨ l = hash destl
            (ShowL_nat x)) ∧ accessStore l v'' = Some (MPointer l') ⟶ l' = l)"
            using STArray[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" srcMem v' v''] a30
by simp
          then show ?thesis using b5 b10 by blast
        qed
      qed
    qed
  qed
next
  case (STValue x)
  then show ?case
  proof intros
    fix l l'
    assume *:"cps2mrec srcl destl (STValue x) srcMem destm = Some v'"
      and **:"(TypedStoSubpref l destl (STValue x) ∨ l = destl) ∧ accessStore l v' = Some (MPointer
l')"
    have "Some v' = (let v = accessStorage x srcl srcMem in Some (updateTypedStore destl (MValue v)
(MTValue x) destm))"
      using cps2mrec.simps(2)[of srcl destl x srcMem destm] * by simp
    then obtain v where b10: "accessStore destl v' = Some (MValue v)"
      unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
    show "l' = l" using b10 ** by auto
  qed
next
  case (STMap k v)
  then show ?case using cps2mrec.simps(3) by force
qed

lemma cpm2mSelfPointers:
  assumes "iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1 = Some
v'"
  shows "(∀l l'. TypedMemSubPref l destl (MTArray x1 t) ∧ accessStore l v' = Some (MPointer l') ⟶ l'
= l)" using assms(1)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x =
Some v'"
      and a20: "(∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer l')
                ⟶ l' = l)"
      and a30: "cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t srcMem v' = Some v''" by
blast
  show ?case
  proof intros
```

```
    fix l l'
    assume *:"TypedMemSubPref l destl (MTArray (Suc x) t) ∧ accessStore l v'' = Some (MPointer l')"
    show "l' = l"
    proof(cases t)
      case (MTArray x11 x12)

      then show ?thesis
      proof(cases "TypedMemSubPref l destl (MTArray x t)")
        case True
        have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using True by (rule
TypedMemSubPref_index_disjoint)
        then have b50:"accessStore l v' = accessStore l v''"
          using * cpm2mrec_SubPrefixes a30 by simp
        then show ?thesis using a20 True * b50 by metis
      next
        case False
        then have "TypedMemSubPref l (hash destl (ShowL_nat x)) t ∨ l = (hash destl (ShowL_nat x))"
using TypedMemSubPref.simps(2)[of l destl "(Suc x)" t] *
          using less_Suc_eq by auto
        then show ?thesis using a30 cpm2mrecPtrsPointToSelf[of "(hash srcl (ShowL_nat x))" "(hash destl
(ShowL_nat x))" t srcMem v' v''] * by blast
      qed
    next
      case (MTValue x2)
      then have "Some v'' = (case accessStore (hash srcl (ShowL_nat x)) srcMem of None ⇒ None
                | Some (MValue v) ⇒ Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v)
(MTValue x2) v')
                | Some (MPointer literal) ⇒ None)"
        using a30 cpm2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 srcMem
v'] by simp
      have "(∃ i<Suc x. TypedMemSubPref l (hash destl (ShowL_nat i)) t ∨ l = hash destl
          (ShowL_nat i))" using TypedMemSubPref.simps(2)[of l destl "Suc x" t] * by auto
      then obtain i where b400:"i< Suc x ∧ TypedMemSubPref l (hash destl (ShowL_nat i)) (MTValue x2)"
using MTValue by auto
      then have "l = (hash destl (ShowL_nat i))" by simp
      then obtain val where "accessStore l v'' = Some (MValue val)" using Cpm2mrec_val_types[of srcl
destl x2 srcMem m "Suc x" v''] 2(2) MTValue b400 by auto
      then show ?thesis using * by simp
    qed
  qed
qed

lemma cps2mSelfPointers:
  assumes "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1 = Some
v'"
  shows "(∀l l'. l ≠ destl ∧ TypedStoSubpref l destl (STArray x1 t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l)" using assms(1)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x =
Some v'"
      and a20: "(∀l l'. l ≠ destl ∧ TypedStoSubpref l destl (STArray x t) ∧ accessStore l v' = Some
(MPointer l')
                ⟶ l' = l)"
      and a30: "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t srcMem v' = Some v''" by
blast
  show ?case
  proof intros
    fix l l'
    assume *:"l ≠ destl ∧ TypedStoSubpref l destl (STArray (Suc x) t) ∧ accessStore l v'' = Some
```

```
(MPointer l')"
    show "l' = l"
    proof(cases t)
      case (STArray x11 x12)

      then show ?thesis
      proof(cases "TypedStoSubpref l destl (STArray x t)")
        case True
        then obtain i where b10:"i<x ∧ (TypedStoSubpref l (hash destl (ShowL_nat i)) t ∨ l = (hash
destl (ShowL_nat i)))"
          using * by auto
        then have b15:"LSubPrefL2 l (hash destl (ShowL_nat i))" using TypedStoSubpref_imp_LSubPrefL2[of
l "(hash destl
            (ShowL_nat i))" t] unfolding LSubPrefL2_def by auto
        then have b20:"(hash destl (ShowL_nat i)) ≠ (hash destl (ShowL_nat x))" using b10
          by (metis Read_Show_nat'_id hash_never_equal_suffix less_not_refl)
        have "¬ LSubPrefL2 l (hash destl (ShowL_nat x))" using LSubPrefL2_def[of l "hash destl
            (ShowL_nat x)"] b15 b20 Mutual_NonSub_SpecificNonSub hash_int_prefix ShowLNatDot by auto
        then have b50:"accessStore l v' = accessStore l v''"
          using * cps2mrec_SubPrefixes a30 by simp
        then show ?thesis using a20 True * b50 by metis
      next
        case False
        then have "TypedStoSubpref l (hash destl (ShowL_nat x)) t ∨ l = (hash destl (ShowL_nat x))"
using TypedStoSubpref.simps(2)[of l destl "(Suc x)" t] *
          using less_Suc_eq by auto
        then show ?thesis using a30 cps2mrecPtrsPointToSelf[of "(hash srcl (ShowL_nat x))" "(hash destl
(ShowL_nat x))" t srcMem v' v''] * by blast
      qed
    next
      case (STValue x2)
      then have "Some v'' = (let v = accessStorage x2 (hash srcl (ShowL_nat x)) srcMem in Some
(updateTypedStore (hash
          destl (ShowL_nat x)) (MValue v) (MTValue x2) v'))"
        using a30 cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 srcMem
v'] by simp
      have "(∃ i<Suc x. TypedStoSubpref l (hash destl (ShowL_nat i)) t ∨ l = hash destl
          (ShowL_nat i))" using TypedStoSubpref.simps(2)[of l destl "Suc x" t] * by auto
      then obtain i where b400:"i< Suc x ∧ TypedMemSubPref l (hash destl (ShowL_nat i)) (MTValue x2)"
using STValue by auto
      then have "l = (hash destl (ShowL_nat i))" by simp
      then obtain val where "accessStore l v'' = Some (MValue val)" using Cps2mrec_val_types[of srcl
destl x2 srcMem m "Suc x" v''] 2(2) STValue b400 by auto
      then show ?thesis using * by simp
    next
      case (STMap k v)
      then show ?thesis
        using a30 cps2mrec.simps(3) by auto
    qed
  qed
qed

lemma cpm2mrec_preserves_prior_child_case:
  assumes iter: "iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) m x = Some v'"
    and step: "cpm2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms v' = Some v''"
    and mcon: "MCon (MTArray x t) v' ld"
    and iless: "i < x"
  shows "case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
proof -
  let ?child = "hash ld (ShowL_nat i)"
```

```
  let ?changed = "hash ld (ShowL_nat x)"
  have self_root:
    "∀l l'. TypedMemSubPref l ld (MTArray x t) ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l"
    using cpm2mSelfPointers[OF iter] by blast
  then have self_child:
    "∀l l'. TypedMemSubPref l ?child t ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l"
    using moreSpecificTypedSubPref[of ld x t v'] iless by blast
  have outside:
    "∀destl'. ¬ LSubPrefL2 destl' ?changed ⟶ accessStore destl' v' = accessStore destl' v''"
    using cpm2mrec_SubPrefixes[of "(hash ls (ShowL_nat x))" ?changed t ms v' v''] step by simp
  have neq_hash: "?child ≠ ?changed"
    using iless by (metis Read_Show_nat'_id hash_never_equal_suffix less_irrefl_nat)
  then have not_sub_child: "¬ LSubPrefL2 ?child ?changed"
    using ShowLNatDot hash_int_prefix by simp
  from neq_hash have not_sub_changed: "¬ LSubPrefL2 ?changed ?child"
    using ShowLNatDot hash_int_prefix by simp
  have keep_child:
    "accessStore ?child v' = accessStore ?child v''"
    using outside not_sub_child by simp
  show ?thesis
  proof (cases t)
    case (MTArray len' arr')
    then obtain loc2 where ptr:
      "accessStore ?child v' = Some (MPointer loc2)"
      using MConArrayPointers[of x len' arr' v' ld i] mcon iless by auto
    have loc_eq: "loc2 = ?child"
      using self_root iless ptr by auto
    have child_mcon_v':
      "MCon t v' ?child"
      using mcon ptr loc_eq MTArray iless by auto
    have child_mcon_v'':
      "MCon t v'' ?child"
      using MCon_mem_preserved_disjoint_update[of ?child ?changed t v' v'']
        self_child outside child_mcon_v' not_sub_child not_sub_changed
      by blast
    then show ?thesis
      using keep_child ptr loc_eq MTArray by force
  next
    case (MTValue x2)
    then obtain val where val:
      "accessStore ?child v' = Some (MValue val)"
      using mcon iless
      using MCon_sub_MTVal_imps_val by presburger
    have child_mcon_v':
      "MCon t v' ?child"
      using mcon MTValue iless val by auto
    have child_mcon_v'':
      "MCon t v'' ?child"
      using MCon_mem_preserved_disjoint_update[of ?child ?changed t v' v'']
        self_child outside child_mcon_v' not_sub_child not_sub_changed
      by blast
    then show ?thesis
      using keep_child val MTValue
      by (metis MTValue child_mcon_v'' keep_child val Option.option.simps(5) memoryvalue.simps(5)
mtypes.simps(6))
  qed
qed

lemma cps2mrec_preserves_prior_child_case:
  assumes iter: "iter' (λi. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t' ms) m x = Some
v'"
    and step: "cps2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t' ms v' = Some v''"
    and mcon: "MCon (MTArray x t) v' ld"
    and compat: "cps2mTypeCompatible t' t"
```

```
    and iless: "i < x"
  shows "case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
         | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
         | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
proof -
  let ?child = "hash ld (ShowL_nat i)"
  let ?changed = "hash ld (ShowL_nat x)"
  have self_child:
    "∀ l l'. TypedStoSubpref l ?child t' ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l"
    using cps2mSelfPointers[OF iter] iless
    by (metis <∀ l l'.
   l ≠ ld ∧
   TypedStoSubpref l ld (STArray x t') ∧
   accessStore l v' = Some (MPointer l') ⟶
   l' = l> iless TypedStoSubpref.simps(2) TypedStoSubpref_hashes)
  have outside:
    "∀ destl'. ¬ LSubPrefL2 destl' ?changed ⟶ accessStore destl' v' = accessStore destl' v''"
    using cps2mrec_SubPrefixes[of "(hash ls (ShowL_nat x))" ?changed t' ms v' v''] step by simp
  have neq_hash: "?child ≠ ?changed"
    using iless by (metis Read_Show_nat'_id hash_never_equal_suffix less_irrefl_nat)
  then have not_sub_child: "¬ LSubPrefL2 ?child ?changed"
    using ShowLNatDot hash_int_prefix by simp
  from neq_hash have not_sub_changed: "¬ LSubPrefL2 ?changed ?child"
    using ShowLNatDot hash_int_prefix by simp
  have keep_child:
    "accessStore ?child v' = accessStore ?child v''"
    using outside not_sub_child by simp
  show ?thesis
  proof (cases t)
    case (MTArray len' arr')
    then obtain loc2 where ptr:
      "accessStore ?child v' = Some (MPointer loc2)"
      using MConArrayPointers[of x len' arr' v' ld i] mcon iless by auto
    have loc_eq: "loc2 = ?child"
      using self_child ptr TypedStoSubpref_sameLoc by blast
    have child_mcon_v':
      "MCon t v' ?child"
      using mcon ptr loc_eq MTArray iless by auto
    have child_mcon_v'':
      "MCon t v'' ?child"
      using MCon_preserved_under_disjoint_location[of ?child ?changed t' v' t v'']
        self_child compat outside child_mcon_v' not_sub_child not_sub_changed
      by blast
    then show ?thesis
      using keep_child ptr loc_eq MTArray by force
  next
    case (MTValue x2)
    then obtain val where val:
      "accessStore ?child v' = Some (MValue val)"
      using mcon iless
      by (metis mcon iless MTValue MCon_sub_MTVal_imps_val)
    have child_mcon_v':
      "MCon t v' ?child"
      using mcon MTValue iless val by auto
    have child_mcon_v'':
      "MCon t v'' ?child"
      using MCon_preserved_under_disjoint_location[of ?child ?changed t' v' t v'']
        self_child compat outside child_mcon_v' not_sub_child not_sub_changed
      by blast
    then show ?thesis
      using keep_child val MTValue
      by (metis MTValue val keep_child child_mcon_v'' Option.option.simps(5) memoryvalue.simps(5))
```

```
mtypes.simps(6))
  qed
qed

lemma cpm2srec_preserves_prior_child_scon:
  assumes step: "cpm2srec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms v' = Some v''"
    and scon: "SCon tp (hash ld (ShowL_nat i)) v'"
    and iless: "i < x"
  shows "SCon tp (hash ld (ShowL_nat i)) v''"
proof -
  have not_sub1: "¬ LSubPrefL2 (hash ld (ShowL_nat i)) (hash ld (ShowL_nat x))"
    using iless by (metis ShowLNatDot hash_int_prefix hashesIntSame nat_neq_iff)
  then have not_sub2: "¬ LSubPrefL2 (hash ld (ShowL_nat x)) (hash ld (ShowL_nat i))"
    using iless scon
    by (metis ShowLNatDot hash_int_prefix)
  have same_root:
    "∀t. accessStorage t (hash ld (ShowL_nat i)) v' = accessStorage t (hash ld (ShowL_nat i)) v''"
    using cpm2srec_SubPrefixes step not_sub1 by auto
  have same_deeper:
    "∀i' t. accessStorage t (hash (hash ld (ShowL_nat i)) i') v' = accessStorage t (hash (hash ld
(ShowL_nat i)) i') v''"
    using not_sub1 not_sub2 Mutual_NonSub_SpecificNonSub step cpm2srec_SubPrefixes by blast
  show ?thesis
    using SCon_noDots[OF scon same_deeper same_root ShowLNatDot] .
qed
```

11.7 Consistency Implication Lemmas for Recursive Copy Functions

These are the key correctness lemmas for the single-step recursive copy functions. They state that, given a well-formed source, a successful copy produces a well-formed destination.

- *cpm2mrec_imps_MCon_dest*: copying from a *MCon*-consistent memory region to *destl* makes *destl MCon*-consistent in the result.
- *cps2mrec_SCon_imps_MCon_dest*: copying from a *SCon*-consistent storage region into memory makes the destination *MCon*-consistent.
- *cpm2srec_imps_SCon_dest*: copying from a *MCon* memory region into storage makes the destination *SCon*-consistent.

  The proofs proceed by induction on the type structure, with the pointer and value cases handled separately.

```
lemma cps2mrec_SCon_imps_MCon_dest:
  "SCon tp srcl srcMem ∧ cps2mTypeCompatible (STArray l tp) (MTArray l t') ∧
   cps2mrec srcl destl tp srcMem a = Some v'''
         ⟶ MCon t' v''' destl
"
proof (induction tp arbitrary:srcl destl srcMem a v''' l t')
  case (STArray x1 tp)
  show ?case
  proof intros
    assume " SCon (STArray x1 tp) srcl srcMem ∧ cps2mTypeCompatible (STArray l (STArray x1 tp))
(MTArray l t') ∧ cps2mrec srcl destl (STArray x1 tp) srcMem a = Some v''''"
    then have *:" SCon (STArray x1 tp) srcl srcMem"
      and **:"cps2mTypeCompatible (STArray l (STArray x1 tp)) (MTArray l t')"
      and ***:"cps2mrec srcl destl (STArray x1 tp) srcMem a = Some v''' "
      by auto+
    obtain t''' where t'''def:"t' = MTArray x1 t'''" using **
      by (metis mtypes.exhaust cps2mTypeCompatible.simps(2) cps2mTypeCompatible.simps(3))

    then have a5:"Some v''' = (case cps2mTypeConvert tp of None ⇒ None
     | Some t' ⇒
         let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a
         in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) m x1)"
```

```
    using *** cps2mrec.simps(1)[of srcl destl x1 tp srcMem a] by simp
  then obtain t3 where t'Def:"cps2mTypeConvert tp = Some t3" by fastforce
  then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t3) a" by auto
  then have "MCon (MTArray x1 t''') v''' destl"
  proof(cases "x1>0")
    case True
    then have srcMcon:"(∀ i∈{0..x1 - 1}. SCon tp (hash srcl (ShowL_nat i)) srcMem)"
      using * SCon.simps(2)[of x1 tp srcl srcMem] by auto
    then have **:"cps2mTypeCompatible (STArray x1 tp) (MTArray x1 t''')" using ** t'''def by auto
    then have v''def:"Some v''' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat
i)) tp srcMem) m x1"
      using a5 mdef t'Def
      by (metis (lifting) Option.option.simps(5))
    have "∀ i<x1. case accessStore (hash destl (ShowL_nat i)) v''' of None ⇒ False
          | Some (MValue val) ⇒ (case t''' of MTValue typ ⇒ MCon t''' v''' (hash destl (ShowL_nat
i)) | _ ⇒ False)
          | Some (MPointer loc2) ⇒ (case t''' of MTArray len' arr' ⇒ MCon t''' v''' loc2 | MTValue
Types ⇒ False)"
      using * **
    proof(induction rule: iter'_induct[OF _ _ v''def[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem)
m x = Some v'"
          and a20:"SCon (STArray x tp) srcl srcMem ⟶ cps2mTypeCompatible (STArray x tp) (MTArray x
t''') ⟶
                        (∀ i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                          | Some (MValue val) ⇒ (case t''' of MTValue typ ⇒ MCon t''' v' (hash destl
(ShowL_nat i)) | _ ⇒ False)
                          | Some (MPointer loc2) ⇒ (case t''' of MTArray len' arr' ⇒ MCon t''' v'
loc2 | MTValue Types ⇒ False))"
          and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) tp srcMem v' = Some
v''" by blast
      have "SCon (STArray x tp) srcl srcMem " using 2(3) using SCon.simps by auto
      then have a40:"(∀ i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                          | Some (MValue val) ⇒ (case t''' of MTValue typ ⇒ MCon t''' v' (hash destl
(ShowL_nat i)) | _ ⇒ False)
                          | Some (MPointer loc2) ⇒ (case t''' of MTArray len' arr' ⇒ MCon t''' v'
loc2 | MTValue Types ⇒ False))"
        using a20 2 by auto
      have a50:"(∃ p. accessStore destl v' = Some (MPointer p)) ∨ accessStore destl v' = None" using
2
      proof -
        have "accessStore destl v'' = accessStore destl v'"
          using a30 cps2mrec_preserves_dest_root by (metis)
        moreover have "accessStore destl m = accessStore destl v''"
          using "2.hyps" cps2mAccessPrePost[of v'' srcl destl tp srcMem m "Suc x"] hash_inequality by
auto
        moreover have "accessStore destl m = Some (MPointer destl)" using mdef
          unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def by auto
        ultimately show ?thesis by metis
      qed
      show ?case
      proof intros
        fix i
        assume iless:"i<Suc x"
        show "case accessStore (hash destl (ShowL_nat i)) v'' of None ⇒ False
              | Some (MValue val) ⇒ (case t''' of MTValue x ⇒ MCon t''' v'' (hash destl (ShowL_nat
i)) | _ ⇒ False)
          | Some (MPointer loc2) ⇒ (case t''' of MTArray x xa ⇒ MCon t''' v'' loc2 | MTValue Types
⇒ False)"
```

```
          proof(cases "i < x")
            case True
            then have MCondestl:"MCon (MTArray x t''') v' destl" using a40 MCon.simps(2)[of x t''' v'
destl] a50 by simp
            then show ?thesis
            proof(cases t''')
              case mtr:(MTArray x11 x12)
              then obtain stotp where stotp:"tp = STArray x11 stotp" using 2(4) iless True
cps2mTypeCompatible.simps
                by (metis stypes.exhaust)
              have tpt''comp:"cps2mTypeCompatible tp t'''" using 2(4)by simp

              have "Some v'' =
                    (case cps2mTypeConvert stotp of None ⇒ None
                  | Some t' ⇒
                    let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat
x))) (MTArray x11 t') v'
                    in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i))
                        (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) stotp srcMem) m x11)"
                using cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" " (hash destl (ShowL_nat x))" x11
stotp srcMem v' ] a30 stotp by auto
              then have b10:" (∀destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore
destl' v' = accessStore destl' v'')"
                using STArray(1)[of " (hash srcl (ShowL_nat x))" srcMem ] a30 cps2mrec_SubPrefixes[of "
(hash srcl
                    (ShowL_nat x))" " (hash destl (ShowL_nat x))" tp srcMem v' v''] by simp
              then have b15:"(hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)
                              ∧ ¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))
                              ⟶ accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'')" by simp
              have " (ShowL_nat i) ≠ (ShowL_nat x)" using True
                by (metis Read_Show_nat'_id less_irrefl_nat)
              then have b16:"hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)"
                by (simp add: hash_never_equal_suffix)
              have b17:"¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))" using
hash_int_prefix ShowLNatDot b16 by simp
              have b18:"¬ LSubPrefL2 (hash destl (ShowL_nat x)) (hash destl (ShowL_nat i))" using
ShowLNatDot hash_int_prefix b16 by simp
              have b19:"∀destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore destl'
v' = accessStore destl' v''" using b10 by auto
              have accessSt:"accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v''"
                by (simp add: b15 b16 b17)
              then obtain val where valdef:" accessStore (hash destl (ShowL_nat i)) v' = Some (MPointer
val)"
                using MConArrayPointers[of x x11 x12 v' destl i] a40 MCondestl True mtr by auto
              have "Suc x - 1 = x" by simp
              then have subPref:"TypedStoSubpref (hash destl (ShowL_nat i)) destl (STArray x tp)"
                using 2(3) True
                by (metis True TypedStoSubpref.simps(2) stotp)
              have child_self:"∀l l'. TypedStoSubpref l (hash destl (ShowL_nat i)) tp ∧ accessStore l
v' = Some (MPointer l') ⟶ l' = l"
              proof (intro allI impI)
                fix l l'
                assume asm:"TypedStoSubpref l (hash destl (ShowL_nat i)) tp ∧ accessStore l v' = Some
(MPointer l')"
                have l_neq:"l ≠ destl"
                  using asm TypedStoSubpref_hashes by blast
                have "TypedStoSubpref l destl (STArray x tp)"
                  using asm True TypedStoSubpref.simps(2) by blast
                then show "l' = l"
                  using cps2mSelfPointers[of srcl destl tp srcMem m x v'] a10 asm l_neq by blast
              qed
              have valeq:"val = (hash destl (ShowL_nat i))"
```

```
            using child_self valdef TypedStoSubpref_sameLoc by blast
          then have b20:"MCon t''' v' (hash destl (ShowL_nat i))" using a40 mtr True valdef by
fastforce
          then have "MCon t''' v'' (hash destl (ShowL_nat i))" using MCon_preserved_under_disjoint_location[of
              "(hash destl (ShowL_nat i))" "(hash destl (ShowL_nat x))" tp v' t''' v''] a10 b17
              b18 b20 b19 child_self tpt''comp by blast
          then show ?thesis using accessSt valdef mtr valeq by (metis mtypes.simps(5)
memoryvalue.simps(6) Option.option.simps(5))
        next
          case (MTValue x2)
          then have stotp:"tp = (STValue x2)" using 2(4) iless True cps2mTypeCompatible.simps
            by (metis stypes.exhaust)
          have tpt''comp:"cps2mTypeCompatible tp t''''" using 2(4) by simp
          then have "(let v = accessStorage x2 (hash srcl (ShowL_nat x)) srcMem
                        in Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2)
v')) = Some v''"
            using cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2
srcMem v'] a30 stotp by simp
          then have b10:"(∀ destl'. ¬ LSubPrefL2 destl' (hash destl (ShowL_nat x)) ⟶ accessStore
destl' v' = accessStore destl' v'')"
            using STArray(1)[of " (hash srcl (ShowL_nat x))" srcMem ] a30 cps2mrec_SubPrefixes[of "
(hash srcl
              (ShowL_nat x))" " (hash destl (ShowL_nat x))" tp srcMem v' v''] by simp
          then have b15:"(hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)
                        ∧ ¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))
                        ⟶ accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'')" by simp
          have " (ShowL_nat i) ≠ (ShowL_nat x)" using True
            by (metis Read_Show_nat'_id less_irrefl_nat)
          then have b16:"hash destl (ShowL_nat i) ≠ hash destl (ShowL_nat x)"
            by (simp add: hash_never_equal_suffix)

          have b17:"¬ LSubPrefL2 (hash destl (ShowL_nat i)) (hash destl (ShowL_nat x))" using
ShowLNatDot hash_int_prefix b16 by simp
          have b18:"¬ LSubPrefL2 (hash destl (ShowL_nat x)) (hash destl (ShowL_nat i))" using
ShowLNatDot hash_int_prefix b16 by simp
          have b20:"MCon t''' v' (hash destl (ShowL_nat i))" using a40 MTValue True by (auto
split:option.splits memoryvalue.splits mtypes.splits)
          then have "MCon t''' v'' (hash destl (ShowL_nat i))" using MCon_mem_preserved_disjoint_update[of
"hash
              destl (ShowL_nat i)" "hash destl (ShowL_nat x)" ] using MTValue b10 b17 b18 by
              simp
          moreover have "accessStore (hash destl (ShowL_nat i)) v' = accessStore (hash destl
(ShowL_nat i)) v'"
            by (simp add: b15 b16 b17)
          moreover obtain val where "accessStore (hash destl (ShowL_nat i)) v' = Some (MValue
val)"
            using a40 MTValue True b20 by (auto split:option.splits memoryvalue.splits
mtypes.splits)
          ultimately show ?thesis
            by (simp add: MTValue b15 b16 b17)
        qed


      next
        case False
        then have iIsx:"i = x" using iless by simp
        then show ?thesis
        proof(cases t''')
          case mtr:(MTArray x11 x12)
          then obtain stotp where stotp:"tp = STArray x11 stotp" using 2(4) iless True
cps2mTypeCompatible.simps
            by (metis stypes.exhaust)
          have tpt''comp:"cps2mTypeCompatible tp t''''" using 2(4) by simp
```

```
            have b20:"Some v'' =
                  (case cps2mTypeConvert stotp of None ⇒ None
                | Some t' ⇒
                  let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat
x))) (MTArray x11 t') v'
                  in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i))
                      (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) stotp srcMem) m x11)"
              using cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" " (hash destl (ShowL_nat x))" x11
stotp srcMem v' ] a30 stotp by auto
            then obtain t2 where t2Def:" cps2mTypeConvert stotp = Some t2" by fastforce
            then obtain m where b30:"m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash
destl (ShowL_nat x))) (MTArray x11 t2) v'" by auto
            then have b40:"Some v'' = iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x))
                (ShowL_nat i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i))
                stotp srcMem) m x11"
              using b20 b30 t2Def by (metis (no_types, lifting) Option.option.simps(5))
            then have b50:"accessStore (hash destl (ShowL_nat i)) m = Some (MPointer (hash destl
(ShowL_nat x)))"
              using b30 iIsx unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
            have "∀suffs. hash (hash destl (ShowL_nat x)) suffs ≠ hash destl (ShowL_nat x)" using
hash_inequality by simp
            then have b60:"accessStore (hash destl (ShowL_nat i)) v'' = accessStore (hash destl
(ShowL_nat i)) m "
              using cps2mAccessPrePost[of v'' "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))"
stotp srcMem m x11] b40 iIsx by presburger
            have "MCon t''' v'' (hash destl (ShowL_nat x))" using STArray[of "(hash srcl
                (ShowL_nat x))" srcMem "(Suc x)" t''' " (hash destl (ShowL_nat x))" v' v''] 2(4)
                a30 b20 t'''def 2 by simp
            then show ?thesis using b50 b60 mtr by simp
          next
            case (MTValue x2)
            then have stotp:"tp = (STValue x2)" using 2(4) iless True cps2mTypeCompatible.simps
              by (metis stypes.exhaust)
            have tpt''comp:"cps2mTypeCompatible tp t''''" using 2(4) by simp
            then have b10:"(let v = accessStorage x2 (hash srcl (ShowL_nat x)) srcMem
    in Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2) v')) = Some v''"
              using cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2
srcMem v'] a30 stotp by simp
            then obtain v where b20:"v = accessStorage x2 (hash srcl (ShowL_nat x)) srcMem" by (simp
split:option.splits memoryvalue.splits)
            then have b30:"v'' = (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2)
v')" using b10 by (simp split:option.splits memoryvalue.splits)
            then have b40:"accessStore (hash destl (ShowL_nat i)) v'' = Some (MValue v)"
              using iIsx unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by auto
            have "(∀i∈{0..Suc x - 1}. SCon tp (hash srcl (ShowL_nat i)) srcMem)" using 2(3)
SCon.simps(2)[of "Suc x" tp srcl srcMem ] by auto
            then have b50:"typeCon x2 (accessStorage x2 (hash srcl (ShowL_nat i)) srcMem)" using
SCon.simps(1)[of x2
                "(hash srcl (ShowL_nat i))" srcMem] stotp iIsx by simp
            have b60:"(case srcMem $$ hash srcl (ShowL_nat x) of None ⇒ ival x2 | Some v ⇒ v) = v"
using b20 unfolding accessStorage_def by blast
            then show ?thesis using iIsx MTValue b20 iless MCon.simps(1) b40 b50 by (metis
(full_types, lifting) mtypes.simps(6) memoryvalue.simps(5) Option.option.simps(5))
          qed
        qed
      qed
    qed
    moreover have "(∃p. accessStore destl v''' = Some (MPointer p)) ∨ accessStore destl v''' =
None"
    proof -
      have "accessStore destl m = accessStore destl v''''
```

```
        using cps2mAccessPrePost[of v''' srcl destl tp srcMem m x1]
          accessStore_updateStore hash_inequality mdef v''def by blast
      moreover have "accessStore destl m = Some (MPointer destl)"
        using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
by auto
      ultimately show ?thesis by auto
    qed
    ultimately show ?thesis using MCon.simps(2)[of x1 t''' v''' destl]
      by (simp add: True)
  next
    case False
    then show ?thesis
      using "**" t'''def by auto
  qed

  then show " MCon t' v''' destl" using t'''def by auto
 qed
next
 case (STMap x1 tp)
 then show ?case by simp
next
 case (STValue x)
 then show ?case
 proof intros
  assume a1:" SCon (STValue x) srcl srcMem ∧ cps2mTypeCompatible (STArray l (STValue x)) (MTArray l
t') ∧ cps2mrec srcl destl (STValue x) srcMem a = Some v'''"
  then have *:"SCon (STValue x) srcl srcMem"
    and **:"cps2mTypeCompatible (STArray l (STValue x)) (MTArray l t')"
    and ***:"cps2mrec srcl destl (STValue x) srcMem a = Some v''''" by simp+

  then have a10:"Some v''' = (let v = accessStorage x srcl srcMem in Some (updateTypedStore destl
(MValue v) (MTValue x) a))"
    using cps2mrec.simps(2)[of srcl destl x srcMem a ] by simp

  then obtain v where b20:"v = accessStorage x srcl srcMem" by (simp split:option.splits
memoryvalue.splits)
  then have b30:"v''' = (updateTypedStore destl (MValue v) (MTValue x) a)" using a10 by (simp
split:option.splits memoryvalue.splits)
  then have b40:"accessStore destl v''' = Some (MValue v)" unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def by auto
  then have b50:"typeCon x (accessStorage x srcl srcMem)" using SCon.simps(1)[of x "srcl" srcMem] *
by simp
  have b60:"(case srcMem $$ srcl of None ⇒ ival x | Some v ⇒ v) = v" using b20 unfolding
accessStorage_def by blast
  then have "typeCon x v" using b50 b20 by simp
  then show "MCon t' v''' destl" using STValue b20 MCon.simps(1) b40
    by (metis memoryvalue.simps(5) Option.option.simps(5) a1 cps2mTypeCompatible.simps(1)
cps2mTypeCompatible.simps(2) cps2mTypeCompatible.simps(6) extractType.cases)
 qed
qed


lemma cpm2srec_imps_SCon_dest :
 "(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon t' srcMem l)
 ∧(∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon t' srcMem srcl)
∧ cps2mTypeCompatible (STArray l tp) (MTArray l t') ∧
  cpm2srec srcl destl t' srcMem a = Some v'''
       ⟶ SCon tp destl v''''"
proof (induction t' arbitrary:srcl destl srcMem a v''' l tp)
 case (MTArray x1 t')
 show ?case
 proof intros
  assume "(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x1 t') srcMem l) ∧
   (∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTArray x1 t') srcMem srcl) ∧
```

```
      cps2mTypeCompatible (STArray l tp) (MTArray l (MTArray x1 t')) ∧ cpm2srec srcl destl (MTArray
x1 t') srcMem a = Some v'''"
    then have *:"(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x1 t') srcMem l)"
      and ****:"(∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTArray x1 t') srcMem
srcl)"
      and **:"cps2mTypeCompatible (STArray l tp) (MTArray l (MTArray x1 t'))"
      and ***:"cpm2srec srcl destl (MTArray x1 t') srcMem a = Some v'''"
      by blast+
    obtain t''' where t'''def:"tp = STArray x1 t'''" using **
      by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
    obtain ptr where ptrDef:"accessStore srcl srcMem = Some (MPointer ptr)"
      using *** unfolding cpm2srec.simps by (auto split:option.splits memoryvalue.splits)
    then have a5:"(iter' (λi. cpm2srec (hash ptr (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem) a
x1) = Some v'''"
      using *** unfolding cpm2srec.simps by simp

    then have "SCon (STArray x1 t''') destl v'''"
    proof -

      have **:"cps2mTypeCompatible (STArray x1 t''') (MTArray x1 t')" using ** t'''def by auto
      have "∀ i<x1. SCon t''' (hash destl (ShowL_nat i)) v'''" using * ** ptrDef
      proof(induction rule: iter'_induct[OF _ _ a5])
        case (1 v')
        then show ?case by simp
      next
        case (2 x v'')
        then obtain v'
          where a10:" iter' (λi. cpm2srec (hash ptr (ShowL_nat i)) (hash destl (ShowL_nat i)) t' srcMem)
a x = Some v'"
            and a20:"((∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x t') srcMem
l) ⟶
          cps2mTypeCompatible (STArray x t''') (MTArray x t') ⟶ accessStore srcl srcMem = Some
(MPointer ptr)
          ⟶ (∀ i<x. SCon t''' (hash destl (ShowL_nat i)) v'))"
            and a30:"cpm2srec (hash ptr (ShowL_nat x)) (hash destl (ShowL_nat x)) t' srcMem v' = Some
v''" by blast
        show ?case
        proof intros
          fix i
          assume iless:"i<Suc x"
          show "SCon t''' (hash destl (ShowL_nat i)) v'''"
          proof(cases "i < x")
            case True
            then have scOA:"(∀ i<x. SCon t''' (hash destl (ShowL_nat i)) v')" using a20 True 2 by auto
            then have scO:"SCon t''' (hash destl (ShowL_nat i)) v'" using True by simp
            then show ?thesis
              using cpm2srec_preserves_prior_child_scon[OF a30 scO True] by simp
          next
            case False
            then have "i = x" using iless by auto
            moreover have "(∀ l. accessStore (hash ptr (ShowL_nat x)) srcMem = Some (MPointer l) ⟶
MCon t' srcMem l)"
              using "2.prems"(1) MCon_imps_sub_Mcon ptrDef by blast
            moreover have "(∀ val. accessStore (hash ptr (ShowL_nat x)) srcMem = Some (MValue val)
                        ⟶ MCon t' srcMem (hash ptr (ShowL_nat x)))"
              using "2.prems"(1) MCon_imps_sub_Mcon ptrDef by blast
            ultimately show ?thesis using a30 MTArray[of "(hash ptr (ShowL_nat x))" srcMem _ t'''
"(hash destl (ShowL_nat x))" v' v'']
              using "2.prems"(2) by blast
          qed
        qed
      qed
      then show "SCon (STArray x1 t''') destl v'''" using t'''def by auto
    qed
```

```
    then show "SCon tp destl v''''"
      by (simp add: t'''def)
  qed

next
  case (MTValue x)
  show ?case
  proof intros
    assume **:"(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTValue x) srcMem l) ∧
      (∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTValue x) srcMem srcl) ∧
      cps2mTypeCompatible (STArray l tp) (MTArray l (MTValue x)) ∧ cpm2srec srcl destl (MTValue x)
srcMem a = Some v''''"
    then have *:"cpm2srec srcl destl (MTValue x) srcMem a = Some v''''" by simp
    then obtain v where ptrDef:"accessStore srcl srcMem = Some (MValue v)"
      unfolding cpm2srec.simps by (auto split:option.splits memoryvalue.splits)
    then have v'''Def:"(a(destl $$:= v)) = v''''" using * unfolding cpm2srec.simps by (auto
split:option.splits memoryvalue.splits)
    have mcMem:"MCon (MTValue x) srcMem srcl" using ptrDef ** by blast
    have "cps2mTypeCompatible tp (MTValue x)" using ** unfolding cps2mTypeCompatible.simps by simp
    then have "STValue x = tp" using cps2mTypeCompatible.simps
      using cps2mTypeCompatible.elims(2) by auto
    moreover have "SCon (STValue x) destl v''''" using v'''Def mcMem ptrDef accessStorage_def unfold-
ing SCon.simps MCon.simps by auto
    ultimately show "SCon tp destl v''''" using ** by simp
  qed
qed
```

11.8 Top-Level Location Preservation

Any copy operation, recursive or iterated, preserves the top-level location tag (*Toploc*) of the memory. This is a structural invariant: copies only update sub-locations within the destination region and never relocate the overall memory object.

Results are given for both the single-step recursive functions (*cpm2mrec*, *cps2mrec*) and the iterated wrappers (*cpm2m*, *cps2m*).

```
lemma cpm2mrecTopLocSame:
  shows " cpm2mrec srcl destl tp' srcMem a = Some v''' ⟶ Toploc a = Toploc v''''"
proof( induction tp' arbitrary: srcl destl a v''')
  case (MTArray x1 tp)
  show ?case
  proof intros
    assume *:" cpm2mrec srcl destl (MTArray x1 tp) srcMem a = Some v''''"
    then have a10:"Some v''' = (case accessStore srcl srcMem of None ⇒ None | Some (MValue literal) ⇒
None
              | Some (MPointer l) ⇒
              let m = updateTypedStore destl (MPointer destl) (MTArray x1 tp) a in iter' (λi m'.
cpm2mrec (hash l
                  (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem m') m x1)" using
                  cpm2mrec.simps(1)[of srcl destl x1 tp srcMem a] by simp
    then obtain p where a20:"Some (MPointer p) = accessStore srcl srcMem" by (auto
split:option.splits memoryvalue.splits)
    then have a30:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 tp) a in
iter' (λi m'. cpm2mrec (hash p
      (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem m') m x1)"
      using a10 by (auto split:option.splits memoryvalue.splits)
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 tp) a" by auto
    then have a50:"Toploc a = Toploc m"unfolding updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by simp
    then have a60:"Some v''' = iter' (λi. cpm2mrec (hash p (ShowL_nat i)) (hash destl (ShowL_nat i)) tp
srcMem) m x1" using a10 a20 a30 a40 by metis
    then show "Toploc a =Toploc v''''" using a50
    proof(induction rule: iter'_induct[OF _ _ a60[symmetric]])
      case (1 v')
      then show ?case by simp
```

```
    next
      case (2 x v'')
      then obtain v'
        where b10:" iter' (λi. cpm2mrec (hash p (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) m x
= Some v'"
          and b20:"(Some v' = iter' (λi. cpm2mrec (hash p (ShowL_nat i)) (hash destl
              (ShowL_nat i)) tp srcMem) m x ⟶ Toploc a = Toploc m ⟶ Toploc a = Toploc
              v')"
          and b30: "cpm2mrec (hash p (ShowL_nat x)) (hash destl (ShowL_nat x)) tp srcMem v' = Some v''"
by blast
      then have "Toploc a= Toploc v'" using 2 by simp
      then show ?case using MTArray[of " (hash p (ShowL_nat x))" "(hash destl (ShowL_nat x))" v' v''' ]
b30 using MTArray by auto
    qed
  qed
next
  case (MTValue x)
  then show ?case using cpm2mrec.simps(2)[of srcl destl x srcMem a] unfolding updateTypedStore_def
updateTypeStore_def updateStore_def accessStore_def
    by (auto split:option.splits memoryvalue.splits)
qed

lemma cps2mrecTopLocSame:
  shows " cps2mrec srcl destl tp' srcMem a = Some v''' ⟶ Toploc a = Toploc v'''"
proof( induction tp' arbitrary: srcl destl a v''')
  case (STArray x1 tp)
  show ?case
  proof
    assume *:" cps2mrec srcl destl (STArray x1 tp) srcMem a = Some v''''"
    then have a10:"Some v''' = (case cps2mTypeConvert tp of None ⇒ None
     | Some t' ⇒
         let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a
         in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) m x1)"
      using cps2mrec.simps(1)[of srcl destl x1 tp srcMem a] by simp
    then obtain t' where t'Def:"cps2mTypeConvert tp = Some t'" by fastforce
    then obtain m where a40:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a" by auto
    then have a50:"Toploc a = Toploc m" unfolding updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def by simp
    then have a60:"Some v''' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i))
tp srcMem) m x1"
      using a10 a40 t'Def by (metis (no_types, lifting) Option.option.simps(5))
    then show "Toploc a =Toploc v''''" using a50
    proof(induction rule: iter'_induct[OF _ _ a60[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where b10:" iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) m
x = Some v'"
          and b20:"(Some v' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl
              (ShowL_nat i)) tp srcMem) m x ⟶ Toploc a = Toploc m ⟶ Toploc a = Toploc
              v')"
          and b30: "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) tp srcMem v' = Some
v''" by blast
      then have "Toploc a= Toploc v'" using 2 by simp
      then show ?case using STArray[of " (hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" v' v'''
] b30 using STArray by auto
    qed
  qed
next
  case (STValue x)
  then show ?case using cps2mrec.simps(2)[of srcl destl x srcMem a]
    unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
```

```
    by (auto split:option.splits memoryvalue.splits)
next
  case (STMap k v)
  then show ?case using cps2mrec.simps(3) by simp
qed

lemma cpm2mTopLocSame:
  assumes "iter' (λi m. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
    and "x>0"
  shows "Toploc md = Toploc updM"
proof (rule iter'_invariant[OF assms(1)])
  show "Toploc md = Toploc md" by simp
next
  fix i m m'
  assume "Toploc md = Toploc m"
    and "cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  then show "Toploc md = Toploc m'" using cpm2mrecTopLocSame by auto
qed

lemma cps2mTopLocSame:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "Toploc md = Toploc updM"
proof (rule iter'_invariant[OF assms(1)])
  show "Toploc md = Toploc md" by simp
next
  fix i m m'
  assume "Toploc md = Toploc m"
    and "cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  then show "Toploc md = Toploc m'" using cps2mrecTopLocSame by auto
qed
```

11.9 Iterated Single-Change Lemmas: Location-Prefix Restriction

These lemmas lift the recursive single-change results to the iterated copies *cpm2m*, *cpm2s*. A location that is not a sub-prefix of the destination root (or is the root itself) is unchanged in both the value and type stores after the full iteration.

The proof uses *iter'_invariant*: the property is maintained as an invariant across every step of the iteration, relying on the recursive single-change lemmas from the previous section. Variants cover memory-to-memory (*cpm2m*), memory-to-storage (*cpm2s*), and separate value/type store projections.

```
lemma cpm2mSingleChange:
  assumes "iter' (λi m. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs md = accessStore locs updM"
proof (rule iter'_invariant[OF assms(1)])
  show "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs md = accessStore locs md" by
simp
next
  fix i m m'
  assume IH:"∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs md = accessStore locs m"
    and step:"cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  show "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs md = accessStore locs m'"
  proof intros
    fix locs
    assume *:"¬LSubPrefL2 locs ld ∨ locs = ld"
    then have "accessStore locs md = accessStore locs m" using IH by simp
    moreover have "accessStore locs m = accessStore locs m'"
      using cpm2mrec_SubPrefixes[of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t ms m m']
      using step * Not_Sub_More_Specific by (metis cpm2mrec_preserves_dest_root)
    ultimately show "accessStore locs md = accessStore locs m'" by simp
  qed
qed
```

lemma *cpm2mSingleChange_Typed:*
assumes *"iter' (λi m. cpm2mrec (hash ls (ShowL$_{nat}$ i)) (hash ld (ShowL$_{nat}$ i)) t ms m) md x = Some updM"*
shows *"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs md = accessTypeStore locs updM"*
proof *(rule iter'_invariant[OF assms(1)])*
show *"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs md = accessTypeStore locs md"* **by** *simp*
next
fix *i m m'*
assume *IH:"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs md = accessTypeStore locs m"*
and *step:"cpm2mrec (hash ls (ShowL$_{nat}$ i)) (hash ld (ShowL$_{nat}$ i)) t ms m = Some m'"*
show *"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs md = accessTypeStore locs m'"*
proof *intros*
fix *locs*
assume *∗:"¬LSubPrefL2 locs ld ∨ locs = ld"*
then have *"accessTypeStore locs md = accessTypeStore locs m"* **using** *IH* **by** *simp*
moreover have *"accessTypeStore locs m = accessTypeStore locs m'"*
using *cpm2mrec_SubPrefixes_both[of "(hash ls (ShowL$_{nat}$ i))" "(hash ld (ShowL$_{nat}$ i))" t ms m m']*
using *step ∗ Not_Sub_More_Specific* **by** *(metis cpm2mrec_preserves_dest_root_typed)*
ultimately show *"accessTypeStore locs md = accessTypeStore locs m'"* **by** *simp*
qed
qed

lemma *cpm2sSingleChange:*
assumes *"iter' (λi m. cpm2srec (hash ls (ShowL$_{nat}$ i)) (hash ld (ShowL$_{nat}$ i)) t ms m) md x = Some updM"*
shows *"∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t' locs updM"*
proof *(rule iter'_invariant[OF assms(1)])*
show *"∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t' locs md"* **by** *simp*
next
fix *i m m'*
assume *IH:"∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t' locs m"*
and *step:"cpm2srec (hash ls (ShowL$_{nat}$ i)) (hash ld (ShowL$_{nat}$ i)) t ms m = Some m'"*
show *"∀ t' locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStorage t' locs md = accessStorage t' locs m'"*
proof *intros*
fix *t' locs*
assume *∗:"¬LSubPrefL2 locs ld ∨ locs = ld"*
then have *"accessStorage t' locs md = accessStorage t' locs m"* **using** *IH* **by** *simp*
moreover have *"accessStorage t' locs m = accessStorage t' locs m'"*
using *cpm2srec_SubPrefixes[of "(hash ls (ShowL$_{nat}$ i))" "(hash ld (ShowL$_{nat}$ i))" t ms m m']*
using *step ∗ Not_Sub_More_Specific* **by** *(metis cpm2srec_preserves_dest_root)*
ultimately show *"accessStorage t' locs md = accessStorage t' locs m'"* **by** *simp*
qed
qed

lemma *cps2mSingleChange_both:*
assumes *"iter' (λi m. cps2mrec (hash ls (ShowL$_{nat}$ i)) (hash ld (ShowL$_{nat}$ i)) t ms m) md x = Some updM"*
shows *"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶*
accessStore locs md = accessStore locs updM
∧ accessTypeStore locs md = accessTypeStore locs updM"
proof *(rule iter'_invariant[OF assms(1)])*
show *"∀ locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶*
accessStore locs md = accessStore locs md
∧ accessTypeStore locs md = accessTypeStore locs md"
by *simp*
next

```
  fix i m m'
  assume IH:
    "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶
      accessStore locs md = accessStore locs m
    ∧ accessTypeStore locs md = accessTypeStore locs m"
    and step: "cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  show "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶
      accessStore locs md = accessStore locs m'
    ∧ accessTypeStore locs md = accessTypeStore locs m'"
  proof intros
    fix locs
    assume asm: "¬LSubPrefL2 locs ld ∨ locs = ld"
    then have inv1:
      "accessStore locs md = accessStore locs m
        ∧ accessTypeStore locs md = accessTypeStore locs m"
      using IH by blast
    have inv2:
      "accessStore locs m = accessStore locs m'
        ∧ accessTypeStore locs m = accessTypeStore locs m'"
    proof (cases "locs = ld")
      case True
      then show ?thesis
        using step cps2mrec_preserves_dest_root_both by simp
    next
      case False
      with asm have not_sub: "¬LSubPrefL2 locs ld" by simp
      have not_sub_child: "¬LSubPrefL2 locs (hash ld (ShowL_nat i))"
        using not_sub Not_Sub_More_Specific by simp
      show ?thesis
        using cps2mrec_SubPrefixes_both[
          of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t ms m m'] step not_sub_child
        by blast
    qed
    show "accessStore locs md = accessStore locs m'"
      using inv1 inv2 by simp
    show "accessTypeStore locs md = accessTypeStore locs m'"
      using inv1 inv2 by simp
  qed
qed

lemma cps2mSingleChange:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessStore locs md = accessStore locs updM"
  using cps2mSingleChange_both[OF assms] by blast

lemma cps2mSingleChange_Typed:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀locs. ¬LSubPrefL2 locs ld ∨ locs = ld ⟶ accessTypeStore locs md = accessTypeStore locs
updM"
  using cps2mSingleChange_both[OF assms] by blast
```

11.10 Sub-Prefix Preservation: Typed Membership Condition

A second family of sub-prefix preservation lemmas, using *TypedMemSubPref* and *TypedStoSubpref* instead of the plain *LSubPrefL2* relation. These predicates combine location prefix membership with the array-length information encoded in the type, giving a tighter characterisation of which locations are modified.

A location that is not the destination root and is not a typed sub-prefix of the destination is unchanged in both the value and type stores. Results are given for *cpm2mrec*, *cps2mrec* (recursive), and their *both*-access counterparts.

```
lemma cpm2mrec_dest_case:
  assumes step: "cpm2mrec srcl destl t ms md = Some v''"
    and mcon: "MCon t v'' destl"
```

```
  shows "case accessStore destl v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' destl | _ ⇒ False)
          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
proof (cases t)
  case (MTArray x1 t')
  obtain l where ldef: "accessStore srcl ms = Some (MPointer l)"
    using step cpm2mrec.simps(1)[of srcl destl x1 t' ms md] MTArray unfolding Let_def
    by (auto split: if_splits option.splits memoryvalue.splits)
  let ?m = "updateTypedStore destl (MPointer destl) (MTArray x1 t') md"
  have expanded:
    "Some v'' =
      (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') md
       in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t' ms) m x1)"
    using step ldef MTArray cpm2mrec.simps(1)[of srcl destl x1 t' ms md] unfolding Let_def
    by (auto split: if_splits option.splits memoryvalue.splits)
  have run:
    "iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t' ms) ?m x1 = Some v''"
    using expanded unfolding Let_def by simp
  have keep_store:
    "accessStore destl ?m = accessStore destl v''"
    using cpm2mSingleChange run
    by blast
  have root_ptr:
    "accessStore destl v'' = Some (MPointer destl)"
    using keep_store
    unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
    by simp
  show ?thesis
    using root_ptr mcon MTArray by simp
next
  case (MTValue x)
  then obtain v where vdef: "accessStore srcl ms = Some (MValue v)"
    using step cpm2mrec.simps(2)[of srcl destl x ms md]
    by (auto split: option.splits memoryvalue.splits)
  then have v''def: "v'' = updateTypedStore destl (MValue v) (MTValue x) md"
    using step MTValue cpm2mrec.simps(2)[of srcl destl x ms md] by simp
  then show ?thesis
    using mcon MTValue
    unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
    by simp
qed


lemma cps2mrec_dest_case:
  assumes step: "cps2mrec srcl destl tp ms md = Some v''"
    and mcon: "MCon t v'' destl"
  shows "case accessStore destl v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v'' destl | _ ⇒ False)
          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
proof (cases tp)
  case (STArray x1 tp')
  obtain tt where conv: "cps2mTypeConvert tp' = Some tt"
    using step STArray cps2mrec.simps(1)[of srcl destl x1 tp' ms md]
    by (auto split: option.splits memoryvalue.splits)
  let ?m = "updateTypedStore destl (MPointer destl) (MTArray x1 tt) md"
  have expanded:
    "Some v'' =
      (let m = updateTypedStore destl (MPointer destl) (MTArray x1 tt) md
       in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp' ms) m x1)"
    using step STArray conv by (simp add: cps2mrec.simps(1))
  have run:
    "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) tp' ms) ?m x1 = Some v''"
```

```
    using expanded unfolding Let_def by simp
  have keep_store:
    "accessStore destl ?m = accessStore destl v'''"
    using cps2mSingleChange[OF run] by simp
  have root_ptr:
    "accessStore destl v'' = Some (MPointer destl)"
    using keep_store
    unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
    by simp
  then show ?thesis
    using keep_store mcon STArray
    by (metis mcon root_ptr memoryvalue.simps(6) mtypes.exhaust Option.option.simps(5) mtypes.simps(5)
MCon.simps(1))
next
  case (STMap x1 tp')
  then show ?thesis
    using step cps2mrec.simps(3) by simp
next
  case (STValue x)
  have v''def: "v'' = updateTypedStore destl (MValue (accessStorage x srcl ms)) (MTValue x) md"
    using step STValue cps2mrec.simps(2)[of srcl destl x ms md] by simp
  have root_val:
    "accessStore destl v'' = Some (MValue (accessStorage x srcl ms))"
    using v''def
    unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
    by simp
  then show ?thesis
    using mcon STValue root_val
    by (cases t, auto split: option.splits memoryvalue.splits)
qed


lemma cpm2mrec_SubPrefixes_typedPref_both:
  "cpm2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. destl' ≠ destl ∧ (¬ TypedMemSubPref destl' destl tp)
    ⟶ accessStore destl' a = accessStore destl' v'''
      ∧ accessTypeStore destl' a = accessTypeStore destl' v''')"
proof (induction tp arbitrary: srcl destl srcMem a v''')
  case (MTArray x1 t)
  show ?case
  proof
    assume run: "cpm2mrec srcl destl (MTArray x1 t) srcMem a = Some v''''"
    then obtain l where src_ptr: "accessStore srcl srcMem = Some (MPointer l)"
      using cpm2mrec.simps(1)[of srcl destl x1 t srcMem a]
      by (auto split: if_splits option.splits memoryvalue.splits)
    have expanded:
      "Some v''' =
        (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a
         in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1)"
      using run src_ptr cpm2mrec.simps(1)[of srcl destl x1 t srcMem a] by simp
    then obtain m where mdef: "m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto
    have iter_run:
      "iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x1 = Some v''''"
      using expanded mdef by presburger
    have init:
      "∀destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray 0 t)
        ⟶ accessStore destl' a = accessStore destl' m
          ∧ accessTypeStore destl' a = accessTypeStore destl' m"
      using mdef updateTypedStore_neq_loc by metis
    have step:
      "⋀i m1 m2.
        (∀destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1)
```

```
        ⟹ cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2
        ⟹ (∀ destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray (Suc i) t)
          ⟶ accessStore destl' a = accessStore destl' m2
            ∧ accessTypeStore destl' a = accessTypeStore destl' m2)"
    proof intros
      fix i m1 m2 destl'
      assume IH_iter:
        "∀ destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1"
      assume rec: "cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2"
      assume asm: "destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray (Suc i) t)"
      have asm_prev: "destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray i t)"
        using asm TypedMemSubPref.simps(2) lessI
        by auto
      then have inv1:
        "accessStore destl' a = accessStore destl' m1
          ∧ accessTypeStore destl' a = accessTypeStore destl' m1"
        using IH_iter by blast
      have asm_child:
        "destl' ≠ hash destl (ShowL_nat i) ∧ ¬ TypedMemSubPref destl' (hash destl (ShowL_nat i)) t"
        using asm by (metis TypedMemSubPref.simps(2) lessI)
      have inv2:
        "accessStore destl' m1 = accessStore destl' m2
          ∧ accessTypeStore destl' m1 = accessTypeStore destl' m2"
        using MTArray.IH[of "hash l (ShowL_nat i)" "hash destl (ShowL_nat i)" srcMem m1 m2] rec asm_child
        by blast
      show
        "accessStore destl' a = accessStore destl' m2" using inv1 inv2 by simp
      show "accessTypeStore destl' a = accessTypeStore destl' m2" using inv1 inv2
        by auto
    qed
    have final_inv:
      "∀ destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t)
        ⟶ accessStore destl' a = accessStore destl' v'''
          ∧ accessTypeStore destl' a = accessTypeStore destl' v''''"
      using iter'_indexed_invariant[
        where P = "λi m1. ∀ destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1",
        OF iter_run init step] .
    show "∀ destl'. destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTArray x1 t)
      ⟶ accessStore destl' a = accessStore destl' v'''
        ∧ accessTypeStore destl' a = accessTypeStore destl' v''''"
      using final_inv .
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume run: "cpm2mrec srcl destl (MTValue x) srcMem a = Some v''''"
    assume asm: "destl' ≠ destl ∧ ¬ TypedMemSubPref destl' destl (MTValue x)"
    then obtain v where src_val: "accessStore srcl srcMem = Some (MValue v)"
      using run cpm2mrec.simps(2)[of srcl destl x srcMem a]
      by (auto split: option.splits memoryvalue.splits)
    have asm2:"v''' = updateTypedStore destl (MValue v) (MTValue x) a"
      using run src_val cpm2mrec.simps(2)[of srcl destl x srcMem a] by simp
    then show "accessStore destl' a = accessStore destl' v''''"
      using asm updateTypedStore_neq_loc by metis
    show "accessTypeStore destl' a = accessTypeStore destl' v''''"
      using asm asm2 updateTypedStore_neq_loc by metis
  qed
qed
```

```
lemma cpm2mrec_SubPrefixes_typedPref:
  "cpm2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. destl' ≠ destl ∧ (¬ TypedMemSubPref destl' destl tp) ⟶ accessStore destl' a =
accessStore destl' v''')"
  using cpm2mrec_SubPrefixes_typedPref_both by blast

lemma cpm2mrec_SubPrefixes_typedPref_typed:
  "cpm2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. destl' ≠ destl ∧ (¬ TypedMemSubPref destl' destl tp) ⟶ accessTypeStore destl' a =
accessTypeStore destl' v''')"
  using cpm2mrec_SubPrefixes_typedPref_both by blast

lemma cps2mrec_SubPrefixes_typedPref_both:
  "cps2mrec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl tp ⟶
      accessStore destl' a = accessStore destl' v'''
    ∧ accessTypeStore destl' a = accessTypeStore destl' v''')"
proof (induction tp arbitrary: srcl destl srcMem a v''')
  case (STArray x1 t)
  show ?case
  proof
    assume run: "cps2mrec srcl destl (STArray x1 t) srcMem a = Some v'''"
    have expanded:
      "Some v''' = (case cps2mTypeConvert t of None ⇒ None
       | Some t' ⇒
           let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a
           in iter' (λi m'. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m') m
x1)"
      using run cps2mrec.simps(1)[of srcl destl x1 t srcMem a]
      by (auto split: option.splits memoryvalue.splits)
    then obtain t' where t'def:"cps2mTypeConvert t = Some t'" by fastforce
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a" by auto
    have iter_run0:
      "iter' (λi m'. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m')
        (updateTypedStore destl (MPointer destl) (MTArray x1 t') a) x1 = Some v'''"
      using run t'def unfolding cps2mrec.simps(1) Let_def by (simp)
    have iter_run:
      "iter' (λi m'. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m') m x1 =
Some v'''"
      using iter_run0 mdef by simp
    have init:
      "∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray 0 t)
        ⟶ accessStore destl' a = accessStore destl' m
          ∧ accessTypeStore destl' a = accessTypeStore destl' m"
      using mdef updateTypedStore_neq_loc by metis
    have step:
      "⋀i m1 m2.
        (∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1)
        ⟹ cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2
        ⟹ (∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray (Suc i) t)
          ⟶ accessStore destl' a = accessStore destl' m2
            ∧ accessTypeStore destl' a = accessTypeStore destl' m2)"
    proof intros
      fix i m1 m2 destl'
      assume IH_iter:
        "∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1"
      assume rec: "cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m1 = Some m2"
      assume asm: "destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray (Suc i) t)"
      have asm_prev: "destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray i t)"
```

```
        using asm TypedStoSubpref.simps(2) lessI by auto
      then have inv1:
        "accessStore destl' a = accessStore destl' m1
          ∧ accessTypeStore destl' a = accessTypeStore destl' m1"
        using IH_iter by blast
      have asm_child:
        "destl' ≠ hash destl (ShowL_nat i) ∧ ¬ TypedStoSubpref destl' (hash destl (ShowL_nat i)) t"
        using asm
        using TypedStoSubpref_sameLoc by auto
      have inv2:
        "accessStore destl' m1 = accessStore destl' m2
          ∧ accessTypeStore destl' m1 = accessTypeStore destl' m2"
        using STArray.IH[of "hash srcl (ShowL_nat i)" "hash destl (ShowL_nat i)" srcMem m1 m2] rec
asm_child
        by blast
      show "accessStore destl' a = accessStore destl' m2"
        using inv1 inv2 by simp
      show "accessTypeStore destl' a = accessTypeStore destl' m2"
        using inv1 inv2 by auto
    qed
    have final_inv:
      "∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray x1 t)
        ⟶ accessStore destl' a = accessStore destl' v'''
          ∧ accessTypeStore destl' a = accessTypeStore destl' v''''"
      using iter'_indexed_invariant[
        where P = "λi m1. ∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray i t)
          ⟶ accessStore destl' a = accessStore destl' m1
            ∧ accessTypeStore destl' a = accessTypeStore destl' m1",
        OF iter_run init step] .
    show "∀destl'. destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STArray x1 t)
      ⟶ accessStore destl' a = accessStore destl' v'''
        ∧ accessTypeStore destl' a = accessTypeStore destl' v''''"
      using final_inv .
  qed
next
  case (STValue x)
  show ?case
  proof intros
    fix destl'
    assume run: "cps2mrec srcl destl (STValue x) srcMem a = Some v''''"
    assume asm: "destl' ≠ destl ∧ ¬ TypedStoSubpref destl' destl (STValue x)"
    have result:
      "v''' = updateTypedStore destl (MValue (accessStorage x srcl srcMem)) (MTValue x) a"
      using run cps2mrec.simps(2)[of srcl destl x srcMem a] by simp
    then show "accessStore destl' a = accessStore destl' v''''"
      using asm updateTypedStore_neq_loc by metis
    show "accessTypeStore destl' a = accessTypeStore destl' v''''"
      using asm result updateTypedStore_neq_loc by metis
  qed
next
  case (STMap x1 tp)
  then show ?case using cps2mrec.simps(3) by auto
qed

lemma cpm2srec_SubPrefixes_typedPref:
  "cpm2srec srcl destl tp srcMem a = Some v''' ⟶
  (∀destl' tp' t. destl' ≠ destl ∧ cps2mTypeCompatible tp' tp ∧ ¬ TypedStoSubpref destl' destl tp'
      ⟶ accessStorage t destl' a = accessStorage t destl' v''')"
proof(induction tp arbitrary:srcl destl srcMem a v''' )
  case (MTArray x1 tp)
  show ?case
  proof intros
    fix destl' tp' t
    assume **:"cpm2srec srcl destl (MTArray x1 tp) srcMem a = Some v''''"
```

```
      and ***:"destl' ≠ destl ∧ cps2mTypeCompatible tp' (MTArray x1 tp) ∧ ¬ TypedStoSubpref destl'
destl tp'"
    then obtain l where ldef:"accessStore srcl srcMem = Some (MPointer l)"
      using ** cpm2srec.simps(1)[of srcl destl x1 tp srcMem a ] by (auto split:if_splits option.splits
memoryvalue.splits)
    then have a5:"Some v''' = iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) tp
srcMem) a x1"
      using ** cpm2srec.simps(1)[of srcl destl x1 tp srcMem a ] by (auto split:if_splits option.splits
memoryvalue.splits)
    obtain st where stDef:"tp' = STArray x1 st" using ***
      by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
    then have comp:" cps2mTypeCompatible (STArray x1 st) (MTArray x1 tp)"
      using "***" by auto
    then have sub:"¬ TypedStoSubpref destl' destl (STArray x1 st)" using *** stDef by blast
    show "accessStorage t destl' a = accessStorage t destl' v'''" using comp sub
    proof(induction rule: iter'_induct[OF _ _ a5[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cpm2srec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) tp srcMem) a x =
Some v'"
          and a20:"(cps2mTypeCompatible (STArray x st) (MTArray x tp) ⟶ ¬ TypedStoSubpref destl'
destl (STArray x st)
              ⟶ accessStorage t destl' a = accessStorage t destl' v')"
          and a30:"cpm2srec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) tp srcMem v' = Some v''"
by blast

      then show ?case
      proof(cases tp)
        case (MTArray x11 x12)
        then show ?thesis
        proof(cases "x = 0")
          case True
          then have "a = v'" using a10 by simp
          then show ?thesis
            by (metis "2.prems"(1,2) MTArray.IH True TypedStoSubpref_sameLoc a30
cps2mTypeCompatible.simps(2) TypedStoSubpref.simps(2))
        next
          case False
          then have "accessStorage t destl' a = accessStorage t destl' v'" using a20 2 by simp
          moreover have "accessStorage t destl' v' = accessStorage t destl' v''"
            using MTArray.IH[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat x))" srcMem v' v''] a30
            by (metis "2.prems"(1,2) TypedStoSubpref_sameLoc cps2mTypeCompatible.simps(2) lessI
TypedStoSubpref.simps(2))
          ultimately show ?thesis using MTArray.IH[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat
x))" srcMem v' v''] a30 by auto
        qed
      next
        case (MTValue x2)
        then have a50:"cpm2srec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTValue x2) srcMem v'
= Some v''" using a30 by simp
        then obtain v where vdef: "accessStore (hash l (ShowL_nat x)) srcMem = Some (MValue v)"
          using a30 unfolding cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
        then have a70:"Some v'' = Some (v'(hash destl (ShowL_nat x) $$:= v))" using a50 unfolding
cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
        then show ?thesis
        proof(cases "x = 0")
          case True
          then have "a = v'" using a10 by simp
          then show ?thesis
            using "2.prems"(1,2) MTArray True a30
            by (metis TypedStoSubpref_sameLoc cps2mTypeCompatible.simps(2) TypedStoSubpref.simps(2))
```

```
        next
          case False
          then have "x>0" by blast
          then have "cps2mTypeCompatible (STArray x st) (MTArray x tp)"
            using "2.prems"(1) by auto
          have "accessStorage t destl' a = accessStorage t destl' v'" using a20 False 2 by simp
          moreover have "accessStorage t destl' v' = accessStorage t destl' v''" using a30
            by (metis "2.prems"(2) MTArray TypedStoSubpref_sameLoc ‹cps2mTypeCompatible (STArray x st)
(MTArray x tp)› cps2mTypeCompatible.simps(2) lessI
                TypedStoSubpref.simps(2))
          ultimately show ?thesis using a70 a50 a20 by auto
        qed
      qed
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    fix destl' tp' t
    assume *:"cpm2srec srcl destl (MTValue x) srcMem a = Some v'''"
      and **:"destl' ≠ destl ∧ cps2mTypeCompatible tp' (MTValue x) ∧ ¬ TypedStoSubpref destl' destl
tp'"
    then obtain v where vdef: "accessStore srcl srcMem = Some (MValue v)"
      unfolding cpm2srec.simps by (auto split:if_splits option.splits memoryvalue.splits)
    then have "Some v''' = Some (a(destl $$:= v))" using * unfolding cpm2srec.simps by (auto
split:if_splits option.splits memoryvalue.splits)
    moreover have "destl' ≠ destl" using ** LSubPrefL2_def by simp
    ultimately show "accessStorage t destl' a = accessStorage t destl' v''' " using **
accessStorage_def
      by auto
  qed
qed
```

11.11 Iterated Single-Change Lemmas: Typed Sub-Prefix Restriction

These lemmas strengthen the earlier single-change results by using the typed sub-prefix predicates (*TypedMemSubPref* and *TypedStoSubpref*) rather than the plain location-prefix relation. A location that is not a typed sub-prefix of the destination (taking into account the array length encoded in the type) is guaranteed to be unmodified.

Variants are provided for value store, type store, and combined (*both*) access, and for all three iterated copy directions: memory-to-memory (*cpm2m*), memory-to-storage (*cpm2s*), and storage-to-memory (*cps2m*).

```
lemma cpm2mSingleChange_typedPref:
  assumes "iter' (λi m. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀locs. ¬TypedMemSubPref locs ld (MTArray x t) ⟶ accessStore locs md = accessStore locs
updM"
proof (rule iter'_indexed_invariant[OF assms(1)])
  show "∀locs. ¬TypedMemSubPref locs ld (MTArray 0 t) ⟶ accessStore locs md = accessStore locs md"
by simp
next
  fix i m m'
  assume IH:"∀locs. ¬TypedMemSubPref locs ld (MTArray i t) ⟶ accessStore locs md = accessStore locs
m"
    and step:"cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  show "∀locs. ¬TypedMemSubPref locs ld (MTArray (Suc i) t) ⟶ accessStore locs md = accessStore locs
m'"
  proof intros
    fix locs
    assume *:"¬TypedMemSubPref locs ld (MTArray (Suc i) t)"
    then have "accessStore locs md = accessStore locs m" using IH by auto
    moreover have "accessStore locs m = accessStore locs m'"
      using cpm2mrec_SubPrefixes_typedPref[of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t ms m
```

```
m']
      using * step by auto
    ultimately show "accessStore locs md = accessStore locs m'" by simp
  qed
qed

lemma cpm2mSingleChange_typedPref_typed:
  assumes "iter' (λi m. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀ locs. ¬TypedMemSubPref locs ld (MTArray x t) ∨ locs = ld ⟶ accessTypeStore locs md =
accessTypeStore locs updM"
proof (rule iter'_indexed_invariant[OF assms(1)])
  show "∀ locs. ¬TypedMemSubPref locs ld (MTArray 0 t) ∨ locs = ld ⟶ accessTypeStore locs md =
accessTypeStore locs md" by simp
next
  fix i m m'
  assume IH:"∀ locs. ¬TypedMemSubPref locs ld (MTArray i t) ∨ locs = ld ⟶ accessTypeStore locs md =
accessTypeStore locs m"
    and step:"cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  show "∀ locs. ¬TypedMemSubPref locs ld (MTArray (Suc i) t) ∨ locs = ld ⟶ accessTypeStore locs md =
accessTypeStore locs m'"
  proof intros
    fix locs
    assume *:"¬TypedMemSubPref locs ld (MTArray (Suc i) t) ∨ locs = ld"
    then have "accessTypeStore locs md = accessTypeStore locs m" using IH by auto
    moreover have "accessTypeStore locs m = accessTypeStore locs m'"
      using cpm2mrec_SubPrefixes_typedPref_typed[of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t
ms m m']
      using * step by (metis TypedMemSubPref.simps(2) lessI cpm2mrec_preserves_dest_root_typed)
    ultimately show "accessTypeStore locs md = accessTypeStore locs m'" by simp
  qed
qed


lemma cpm2sSingleChange_typedPref:
  assumes "iter' (λi m. cpm2srec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀ locs t' t''. cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ ld
       ∧ ¬TypedStoSubpref locs ld (STArray x t') ⟶ accessStorage t'' locs md = accessStorage t''
locs updM"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. cpm2srec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md x = Some v'"
      and a20:"(∀ locs t' t''. cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ ld ∧ ¬
TypedStoSubpref
         locs ld (STArray x t') ⟶ accessStorage t'' locs md = accessStorage t'' locs v')"
      and a30:"cpm2srec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms v' = Some v''" by blast
  show ?case
  proof intros
    fix locs t' t''
    assume *:"cps2mTypeCompatible (STArray (Suc x) t') (MTArray (Suc x) t) ∧ locs ≠ ld ∧ ¬
TypedStoSubpref locs ld (STArray (Suc x) t')"
    then show "accessStorage t'' locs md = accessStorage t'' locs v''"
    proof(cases "x = 0")
      case True
      then have "accessStorage t'' locs md = accessStorage t'' locs v'" using a10 by simp
      then show ?thesis using cpm2srec_SubPrefixes_typedPref a30
        using "*"
        by (metis True TypedStoSubpref_sameLoc cps2mTypeCompatible.simps(2) TypedStoSubpref.simps(2))
    next
```

```
      case False
      then have "cps2mTypeCompatible (STArray x t') (MTArray x t)"
        by (meson "*" bot_nat_0.not_eq_extremum cps2mTypeCompatible.simps(2))
      have "¬ TypedStoSubpref locs ld (STArray x t')" using False * by simp
      have "¬ TypedStoSubpref locs ld (STArray x t') ⟶ accessStorage t'' locs md = accessStorage t''
locs v'" using a20
        using "*" ‹cps2mTypeCompatible (STArray x t') (MTArray x t)› by blast
      then show ?thesis using a30 cpm2srec_SubPrefixes_typedPref *
        by (metis TypedStoSubpref_sameLoc ‹¬ TypedStoSubpref locs ld (STArray x t')›
cps2mTypeCompatible.simps(2) lessI TypedStoSubpref.simps(2))
    qed

  qed
qed

lemma cps2mSingleChange_typedPref_both:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray x t) ⟶
      accessStore locs md = accessStore locs updM
    ∧ accessTypeStore locs md = accessTypeStore locs updM"
proof (rule iter'_indexed_invariant[OF assms(1)])
  show "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray 0 t) ⟶
      accessStore locs md = accessStore locs md
    ∧ accessTypeStore locs md = accessTypeStore locs md"
    by simp
next
  fix i m m'
  assume IH:
    "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray i t) ⟶
      accessStore locs md = accessStore locs m
    ∧ accessTypeStore locs md = accessTypeStore locs m"
    and step: "cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m = Some m'"
  show "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray (Suc i) t) ⟶
      accessStore locs md = accessStore locs m'
    ∧ accessTypeStore locs md = accessTypeStore locs m'"
  proof intros
    fix locs
    assume asm: "locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray (Suc i) t)"
    have asm_prev: "locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray i t)"
      using asm TypedStoSubpref.simps(2) lessI by auto
    then have inv1:
      "accessStore locs md = accessStore locs m
        ∧ accessTypeStore locs md = accessTypeStore locs m"
      using IH by blast
    have asm_child:
      "locs ≠ hash ld (ShowL_nat i) ∧ ¬TypedStoSubpref locs (hash ld (ShowL_nat i)) t"
      using asm
      by (metis asm TypedStoSubpref.simps(2) TypedStoSubpref_sameLoc lessI)
    have inv2:
      "accessStore locs m = accessStore locs m'
        ∧ accessTypeStore locs m = accessTypeStore locs m'"
      using cps2mrec_SubPrefixes_typedPref_both[
        of "(hash ls (ShowL_nat i))" "(hash ld (ShowL_nat i))" t ms m m'] step asm_child
      by blast
    show "accessStore locs md = accessStore locs m'"
      using inv1 inv2 by simp
    show "accessTypeStore locs md = accessTypeStore locs m'"
      using inv1 inv2 by simp
  qed
qed

lemma cps2mSingleChange_typedPref:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
```

```
updM"
  shows "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray x t) ⟶ accessStore locs md =
accessStore locs updM"
  using cps2mSingleChange_typedPref_both[OF assms] by blast

lemma cps2mSingleChange_typedPref_typed:
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
  shows "∀ locs. locs ≠ ld ∧ ¬TypedStoSubpref locs ld (STArray x t) ⟶ accessTypeStore locs md =
accessTypeStore locs updM"
  using cps2mSingleChange_typedPref_both[OF assms] by blast
```

11.12 Memory and Storage Consistency Propagation Under Iterated Copies

These are the main correctness lemmas for the iterated copy operations. They establish that copying a well-formed (`MCon` or `SCon`) source array into a destination produces a well-formed destination array.

The inductive argument follows the `iter'` structure: at each step the destination has been initialised up to index `len`, the next call to `cpm2mrec` (or `cps2mrec`) writes index `len`, and we must show `MCon` up to `len`. The source is assumed to be `MCon` for the full length throughout.

Also included are helper lemmas about locations that are not typed sub-prefixes of the changed region (`cpm2mNotSubPrefsOfOriginal`, `cpm2mLocationRelatedToTopNotInOrig`, `cps2mNotSubPrefsOfChanged`), which support the inductive step.

```
lemma cpm2mrec_imps_MCon_dest :
  "(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon tp srcMem l)
  ∧(∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon tp srcMem srcl) ∧
   cpm2mrec srcl destl tp srcMem a = Some v'''
         ⟶ MCon tp v''' destl"
proof (induction tp arbitrary:srcl destl srcMem a v''')
  case (MTArray x1 t)
  show ?case
  proof intros
    assume "(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x1 t) srcMem l)
        ∧ (∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTArray x1 t) srcMem srcl)
        ∧ cpm2mrec srcl destl (MTArray x1 t) srcMem a = Some v''''"
    then have *:"(∀ l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x1 t) srcMem l)"
      and **:"cpm2mrec srcl destl (MTArray x1 t) srcMem a = Some v''''"
      and ***:"(∀ val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTArray x1 t) srcMem srcl)
"
      by auto+

    obtain l where ldef:"accessStore srcl srcMem = Some (MPointer l)"
      using ** cpm2mrec.simps(1)[of srcl destl x1 t srcMem a ]
      by (auto split:if_splits option.splits memoryvalue.splits)
    then have a5:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a
                in iter' (λi m'. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem m')
m x1)"
      using ** by (simp add: cpm2mrec.simps(1))
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a" by auto

    then show " MCon (MTArray x1 t) v''' destl"
    proof(cases "x1>0")
      case True
      then have "MCon (MTArray x1 t) srcMem l" using ldef * by auto
      then have srcMcon:"∀ i<x1.
              case accessStore (hash l (ShowL_nat i)) srcMem of None ⇒ False
              | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t srcMem (hash l (ShowL_nat i)) | _ ⇒
False)
              | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t srcMem loc2 | MTValue Types
⇒ False)"
        using MCon.simps(2)[of x1 t srcMem l] by auto

      have v''def:"Some v''' = iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t
srcMem) m x1"
```

```
        using a5 ldef mdef by metis

      have "∀i<x1. case accessStore (hash destl (ShowL_nat i)) v''' of None ⇒ False
            | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v''' (hash destl (ShowL_nat i)) | _
⇒ False)
            | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v''' loc2 | MTValue Types
⇒ False)"
        using * ldef
      proof(induction rule: iter'_induct[OF _ _ v''def[symmetric]])
        case (1 v')
        then show ?case by simp
      next
        case (2 x v'')
        then obtain v'
          where a10:"iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t srcMem) m x
= Some v'"
            and a20:"(∀l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTArray x t) srcMem l)
⟶
                        (∀i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v' (hash destl
(ShowL_nat i)) | _ ⇒ False)
                          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 |
MTValue Types ⇒ False))"
            and a30:"cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) t srcMem v' = Some v''"
by blast
        then have a40:"(∀i<x. case accessStore (hash destl (ShowL_nat i)) v' of None ⇒ False
                          | Some (MValue val) ⇒ (case t of MTValue typ ⇒ MCon t v' (hash destl
(ShowL_nat i)) | _ ⇒ False)
                          | Some (MPointer loc2) ⇒ (case t of MTArray len' arr' ⇒ MCon t v' loc2 |
MTValue Types ⇒ False))" using 2 by fastforce
        have a50:"MCon (MTArray (Suc x) t) srcMem l" using 2 by auto
        have a55:"(∃p. accessStore destl v' = Some (MPointer p)) ∨ accessStore destl v' = None" using
2
        proof -
          have "accessStore destl v'' = accessStore destl v'"
            using a30 cpm2mrec_preserves_dest_root[of l _ destl _ t srcMem v' v''] by blast
          moreover have "accessStore destl m = accessStore destl v''"
            using "2.hyps" cpm2m_iter_dest_unchanged[of v'' l destl t srcMem m "Suc x"]
              accessStore_updateStore hash_inequality mdef by auto
          moreover have "accessStore destl m = Some (MPointer destl)"
            using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
          ultimately show ?thesis by auto
        qed

        show ?case
        proof intros
          fix i
          assume iless:"i<Suc x"

          show "case accessStore (hash destl (ShowL_nat i)) v'' of None ⇒ False
                | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash destl (ShowL_nat i)) | _
⇒ False)
            | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
          proof(cases "i < x")
            case True

            then have MCondestl:"MCon (MTArray x t) v' destl" using a40 MCon.simps(2)[of x t v' destl]
a55 by simp

            then show ?thesis
              using cpm2mrec_preserves_prior_child_case[OF a10 a30 MCondestl True] by simp
```

```
          next
            case False
            then have iIsx:"i = x" using iless by simp
            have child_ptr:
              "⋀loc2. accessStore (hash l (ShowL_nat x)) srcMem = Some (MPointer loc2) ⟹ MCon t srcMem
loc2"
              using a50 using MCon_imps_sub_Mcon by blast
            have child_val:
              "⋀val. accessStore (hash l (ShowL_nat x)) srcMem = Some (MValue val) ⟹ MCon t srcMem
(hash l (ShowL_nat x))"
              using a50 using MCon_imps_sub_Mcon by blast
            have MCON: "MCon t v'' (hash destl (ShowL_nat x))"
              using MTArray[of "(hash l (ShowL_nat x))" srcMem "(hash destl (ShowL_nat x))" v' v'']
                a30 child_ptr child_val
              by blast
            then show ?thesis
              using cpm2mrec_dest_case[OF a30 MCON] iIsx by blast
          qed
        qed
      qed

      moreover have "(∃p. accessStore destl v''' = Some (MPointer p)) ∨ accessStore destl v''' =
None"
      proof -
        have "accessStore destl m = accessStore destl v''''"
          using cpm2m_iter_dest_unchanged[of v''' l destl t srcMem m x1]
            accessStore_updateStore hash_inequality mdef v''def by blast
        moreover have "accessStore destl m = Some (MPointer destl)"
          using mdef unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
by auto
        ultimately show ?thesis by auto
      qed

      ultimately show ?thesis using MCon.simps(2)[of x1 t v''' destl] by (simp add: True)
    next
      case False
      then show ?thesis
        using "*" ldef by auto
    qed
  qed
next
  case (MTValue x)
  then show ?case
  proof intros
    assume "(∀l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTValue x) srcMem l) ∧
            (∀val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTValue x) srcMem srcl)
            ∧ cpm2mrec srcl destl (MTValue x) srcMem a = Some v''''"
    then have *:"(∀l. accessStore srcl srcMem = Some (MPointer l) ⟶ MCon (MTValue x) srcMem l)"
      and **:"(∀val. accessStore srcl srcMem = Some (MValue val) ⟶ MCon (MTValue x) srcMem srcl)"
      and ***:"cpm2mrec srcl destl (MTValue x) srcMem a = Some v''''" by simp+

    then have a10:"Some v''' = (case accessStore srcl srcMem of None ⇒ None
              | Some (MValue v) ⇒ Some (updateTypedStore destl (MValue v) (MTValue x) a)
              | Some (MPointer literal) ⇒ None)"
      using cpm2mrec.simps(2)[of srcl destl x srcMem a ] by simp
    then obtain v where vdef:"Some (MValue v) = accessStore srcl srcMem"
      by (simp split:option.splits memoryvalue.splits)
    then have "accessStore destl v''' = accessStore srcl srcMem"
      using a10 unfolding updateTypedStore_def updateTypeStore_def updateStore_def accessStore_def
      by (auto split:option.splits memoryvalue.splits)
    moreover have "typeCon x v" using vdef ** by simp
    ultimately show "MCon (MTValue x) v''' destl" using ** MCon.simps(1)[of x v''' destl] vdef a10
      by (auto split:option.splits memoryvalue.splits)
```

```
  qed
qed

lemma MCon_cpm2m:
  assumes "iter' (λi m. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
    and "MCon (MTArray x t) ms ls"
    and "(∃p. accessStore (ld) md = Some (MPointer p)) ∨ accessStore (ld) md = None"
  shows " MCon (MTArray x t) updM ld" using assms(2)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case using MCon.simps by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. cpm2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md x = Some v'"
      and a20:"(MCon (MTArray x t) ms ls ∧ 0 < x ⟶ MCon (MTArray x t) v' ld)"
      and a30:"cpm2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms v' = Some v''" by blast
  then have a40:"MCon (MTArray (Suc x) t) ms ls" using "2.prems" by auto


  have " (∀destl'. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStore destl' v' = accessStore
destl' v'')"
    using cpm2mrec_SubPrefixes[of "(hash ls (ShowL_nat x))" "(hash ld (ShowL_nat x))" t ms v' v''] using
a30 by simp
  then have "accessStore ld v' = accessStore ld v''"
    using a30 cpm2mrec_preserves_dest_root by auto


  then have ld:"(∃p. accessStore (ld) v'' = Some (MPointer p)) ∨ accessStore (ld) v'' = None"
  proof(cases "x =0")
    case True
    then have v'MD:"v' = md" using a10 by auto
    then have "cpm2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms md = Some v''" using a30 by
blast
    then have "(∀destl'. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStore destl' md =
accessStore destl' v'')"
      using cpm2mrec_SubPrefixes[of "(hash ls (ShowL_nat x))" "(hash ld (ShowL_nat x))" t ms md v''] by
simp
    then have "accessStore ld md = accessStore ld v''"
      using cpm2mrec_preserves_dest_root v'MD using ‹accessStore ld v' = accessStore ld v''› by blast
    then show ?thesis using assms(3) by simp
  next
    case False
    then have "MCon (MTArray x t) ms ls" using a40 by auto
    then have "MCon (MTArray x t) v' ld" using a20 False by auto
    then show ?thesis
      by (metis ‹accessStore ld v' = accessStore ld v''› MCon.simps(2))
  qed


  have "∀i<Suc x.
            case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
             | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
             | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
  proof intros
    fix i assume *:"i<Suc x"
    show "case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
          | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
```

```
    proof(cases t)
      case (MTArray x11 x12)
      then show ?thesis
      proof(cases "i<x")
        case True
        have b10: "MCon (MTArray x t) v' ld" using a20 a40 True by fastforce
        then show ?thesis
          using cpm2mrec_preserves_prior_child_case[OF a10 a30 b10 True] by simp
      next
        case False
        then have iIsx:"i = x" using * by auto
        have child_ptr:
          "⋀loc2. accessStore (hash ls (ShowL_nat x)) ms = Some (MPointer loc2) ⟹ MCon t ms loc2"
          using a40
          using MCon_imps_sub_Mcon by blast
        have child_val:
          "⋀val. accessStore (hash ls (ShowL_nat x)) ms = Some (MValue val) ⟹ MCon t ms (hash ls
(ShowL_nat x))"
          using a40 using MCon_imps_sub_Mcon by blast
        have MCON:"MCon t v'' (hash ld (ShowL_nat x))"
          using cpm2mrec_imps_MCon_dest[of "(hash ls (ShowL_nat x))" ms t "(hash ld (ShowL_nat x))" v'
v'']
            a30 child_ptr child_val
          by blast
        then show ?thesis
          using cpm2mrec_dest_case[OF a30 MCON] iIsx
          by blast
      qed
    next
      case (MTValue x2)
      then show ?thesis
      proof(cases "i<x")
        case True

        have b10: "MCon (MTArray x t) v' ld" using a20 2 True by auto
        then show ?thesis
          using cpm2mrec_preserves_prior_child_case[OF a10 a30 b10 True] by simp
      next
        case False

        then have iIsx: "i = x" using * by simp
        have child_ptr:
          "⋀loc2. accessStore (hash ls (ShowL_nat x)) ms = Some (MPointer loc2) ⟹ MCon t ms loc2"
          using a40 using MCon_imps_sub_Mcon by blast
        have child_val:
          "⋀val. accessStore (hash ls (ShowL_nat x)) ms = Some (MValue val) ⟹ MCon t ms (hash ls
(ShowL_nat x))"
          using a40 using MCon_imps_sub_Mcon by blast
        have MCON:"MCon t v'' (hash ld (ShowL_nat x))"
          using cpm2mrec_imps_MCon_dest[of "(hash ls (ShowL_nat x))" ms t "(hash ld (ShowL_nat x))" v'
v'']
            a30 child_ptr child_val
          by blast
        then show ?thesis
          using cpm2mrec_dest_case[OF a30 MCON] iIsx
          by blast
      qed
    qed
  qed
  then show ?case using cpm2mrec_imps_MCon_dest MCon.simps(2)[of "Suc x" t v'' "(hash ld (ShowL_nat
x))"] ld by simp
qed

lemma cps2m:
```

```
  assumes "iter' (λi m. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t' ms m) md x = Some
updM"
    and "SCon (STArray x t') ls ms"
    and "cps2mTypeCompatible (STArray x t') (MTArray x t)"
    and "(∃p. accessStore (ld) md = Some (MPointer p)) ∨ accessStore (ld) md = None"
  shows "MCon (MTArray x t) updM ld" using assms(2) assms(3) assms(4)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case using MCon.simps by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. cps2mrec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t' ms) md x = Some v'"
      and a20:"(SCon (STArray x t') ls ms ⟶ 0 < x ⟶ cps2mTypeCompatible (STArray x t') (MTArray x
t) ⟶ MCon (MTArray x t) v' ld)"
      and a30:"cps2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t' ms v' = Some v''" by blast
  then have a40:"SCon (STArray (Suc x) t') ls ms" using "2.prems" by auto

  have " (∀destl'. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStore destl' v' = accessStore
destl' v'')"
    using cps2mrec_SubPrefixes a30 by simp
  then have "accessStore ld v' = accessStore ld v''"
    using a30 cps2mrec_preserves_dest_root by auto
  then have ld:"(∃p. accessStore (ld) v'' = Some (MPointer p)) ∨ accessStore (ld) v'' = None"
  proof(cases "x =0")
    case True
    then have v'MD:"v' = md" using a10 by auto
    then have "cps2mrec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t' ms md = Some v''" using a30
by blast
    then have "(∀destl'. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStore destl' md =
accessStore destl' v'')"
      using cps2mrec_SubPrefixes[of "(hash ls (ShowL_nat x))" "(hash ld (ShowL_nat x))" t' ms md v''] by
simp
    then have "accessStore ld md = accessStore ld v''"
      using cpm2mrec_preserves_dest_root v'MD using ‹accessStore ld v' = accessStore ld v''› by blast
    then show ?thesis using assms(4) by simp
  next
    case False
    then have "SCon (STArray x t') ls ms" using a40 by auto
    then have "MCon (MTArray x t) v' ld" using a20 False
      by (metis assms(3) cps2mTypeCompatible.simps(2) less_nat_zero_code linorder_neqE_nat)
    then show ?thesis
      by (metis ‹accessStore ld v' = accessStore ld v''› MCon.simps(2))
  qed

  have "∀i<Suc x.
            case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
            | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
            | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
  proof intros
    fix i assume *:"i<Suc x"
    show "case accessStore (hash ld (ShowL_nat i)) v'' of None ⇒ False
          | Some (MValue val) ⇒ (case t of MTValue x ⇒ MCon t v'' (hash ld (ShowL_nat i)) | _ ⇒
False)
          | Some (MPointer loc2) ⇒ (case t of MTArray x xa ⇒ MCon t v'' loc2 | MTValue Types ⇒
False)"
    proof(cases t)
      case (MTArray x11 x12)
      then show ?thesis
      proof(cases "i<x")
        case True
        have b10: "MCon (MTArray x t) v' ld" using a20 a40 2(4) 2(5) True by fastforce
```

```
      have b20: "cps2mTypeCompatible t' t" using 2(4) cps2mTypeCompatible.simps True by simp
      then show ?thesis
        using cps2mrec_preserves_prior_child_case[OF a10 a30 b10 b20 True] by simp
    next
      case False
      then have iIsx:"i = x" using * by auto
      have b10:"(∀ i∈{0..Suc x - 1}. SCon t' (hash ls (ShowL_nat i)) ms)"
        using a40 using SCon.simps(2)[of "Suc x" t' "ls" "ms"] by auto
      have MCON:"MCon t v'' (hash ld (ShowL_nat x))"
        using cps2mrec_SCon_imps_MCon_dest[of t' "(hash ls (ShowL_nat x))" ms "Suc x" t "(hash ld
(ShowL_nat x))" v' v'']
          2(4) a30 iIsx b10
        by simp
      then show ?thesis
        using cps2mrec_dest_case[OF a30 MCON] iIsx
        by blast
    qed
  next
    case (MTValue x2)
    then show ?thesis
    proof(cases "i<x")
      case True
      have b10: "MCon (MTArray x t) v' ld" using a20 2 True by auto
      have b20: "cps2mTypeCompatible t' t" using 2(4) cps2mTypeCompatible.simps True by simp
      then show ?thesis
        using cps2mrec_preserves_prior_child_case[OF a10 a30 b10 b20 True] by simp
    next
      case False
      then have iIsx: "i = x" using * by simp
      then have b10:"(∀ i∈{0..Suc x - 1}. SCon t' (hash ls (ShowL_nat i)) ms)"
        using a40 using SCon.simps(2)[of "Suc x" t' "ls" "ms"] by auto
      then have MCON:"MCon t v'' (hash ld (ShowL_nat x))"
        using cps2mrec_SCon_imps_MCon_dest[of t' "(hash ls (ShowL_nat x))" ms "Suc x" t "(hash ld
(ShowL_nat x))" v' v''] 2(4) a30 iIsx b10 by simp
      then show ?thesis
        using cps2mrec_dest_case[OF a30 MCON] iIsx
        by blast
    qed
  qed
  then show ?case using cpm2mrec_imps_MCon_dest MCon.simps(2)[of "Suc x" t v'' "(hash ld (ShowL_nat
x))"] ld by simp
qed


lemma MCon_cpm2s:
  assumes "iter' (λi m. cpm2srec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms m) md x = Some
updM"
    and "MCon (MTArray x t) ms ls"
    and "cps2mTypeCompatible (STArray x t') (MTArray x t)"
  shows "SCon (STArray x t') ld updM" using assms(2) assms(3)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1 v')
  then show ?case using MCon.simps by simp
next
  case (2 x v'')
  then obtain v'
    where a10:"iter' (λi. cpm2srec (hash ls (ShowL_nat i)) (hash ld (ShowL_nat i)) t ms) md x = Some v'"
      and a20:"(MCon (MTArray x t) ms ls ⟶ cps2mTypeCompatible (STArray x t') (MTArray x t) ⟶ SCon
(STArray x t') ld v')"
      and a30:"cpm2srec (hash ls (ShowL_nat x)) (hash ld (ShowL_nat x)) t ms v' = Some v''" by blast
  then have a40:"MCon (MTArray (Suc x) t) ms ls" using "2.prems" by auto

  have " (∀ destl' t. ¬ LSubPrefL2 destl' (hash ld (ShowL_nat x)) ⟶ accessStorage t destl' v' =
```

```
accessStorage t destl' v'')"
    using cpm2srec_SubPrefixes a30 by simp
  then have "∀t. accessStorage t ld v' = accessStorage t ld v''"
    using a30 cpm2srec_preserves_dest_root by auto
  have "(∀i<Suc x. SCon t' (hash ld (ShowL_nat i)) v'')"
  proof intros
    fix i assume *:"i<Suc x"
    show "SCon t' (hash ld (ShowL_nat i)) v''"
    proof(cases t')
      case (STArray x11 x12)
      then show ?thesis
      proof(cases "i<x")
        case True
        have b10: "SCon (STArray x t') ld v' " using a20 a40 2(4)
          using True by fastforce
        then show ?thesis
          using cpm2srec_preserves_prior_child_scon a30 b10 True by simp

      next
        case False
        then have iIsx:"i = x" using * by auto
        then have MCON:"SCon t' (hash ld (ShowL_nat x)) v''"
          using cpm2srec_imps_SCon_dest[of "(hash ls (ShowL_nat x))" ms t _ t' "(hash ld (ShowL_nat x))"
v' v''] 2(4) a30 iIsx
          using MCon_imps_sub_Mcon a40 by blast
        then show ?thesis using MCON iIsx by simp
      qed
    next
      case (STMap x21 x22)
      then show ?thesis using 2 by simp
    next
      case (STValue x3)
      then show ?thesis
      proof(cases "i<x")
        case True
        then have "SCon (STArray x t') ld v'" using a20 a40 2(4) by auto
        then have "SCon t' (hash ld (ShowL_nat i)) v'" using True by simp
        then show ?thesis
          using cpm2srec_preserves_prior_child_scon[OF a30 _ True] STValue
          by blast
      next
        case False
        then have "i = x"
          using "*" by auto
        then show ?thesis
          by (meson MCon_imps_sub_Mcon
              a10 a20 a30
              a40 assms(3) cpm2srec_imps_SCon_dest lessI)
      qed
    qed
  qed
  then show ?case using cpm2mrec_imps_MCon_dest SCon.simps(2)[of "Suc x" t' ld v''] by simp
qed


lemma cpm2mNotSubPrefsOfOriginal:
  assumes "MCon (MTArray x11 x12) m' stl2"
    and "∀l'. l' ≠ stl1 ⟶ ¬ LSubPrefL2 stl1 l'"
    and "∀loc. LSubPrefL2 loc stl1 ⟶ accessStore loc mem' = None"
    and "¬ LSubPrefL2 stl2 stl1"
    and "MCon (MTArray x11 x12) mem' stl2"
    and "∀locs. ¬ LSubPrefL2 locs stl1 ⟶ accessStore locs mem' = accessStore locs m'"
    and "∀l' ≠ stl1. ¬(LSubPrefL2 stl1 l')"
  shows "¬TypedMemSubPrefPtrs m' x11 x12 stl2 stl1" using assms
```

```
proof(induction x12 arbitrary:stl2 x11)
  case (MTArray x1 x12)
  have "(∀i<x11. ∃l. accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l) ∧ (l ≠ stl1 ∧
¬TypedMemSubPrefPtrs m' x1 x12 l stl1))"
  proof intros
    fix i
    assume idef':"i<x11"
    have d5:"∀i<x11. accessStore (hash stl2 (ShowL_nat i)) m' = accessStore (hash stl2 (ShowL_nat i))
mem'" using MTArray.prems LSubPrefL2_def
      by (metis Mutual_NonSub_SpecificNonSub)
    then have d10:"∀i<x11. ∃l'. accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l')" using
MTArray.prems
      by (metis MConArrayPointers less_nat_zero_code neq0_conv)

    then obtain l' where idef:"accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l')" using
MTArray.prems(1) idef' by fastforce
    then have idef2:" accessStore (hash stl2 (ShowL_nat i)) mem' = Some (MPointer l')" using d5 idef'
by auto

    have "(case accessStore (hash stl2 (ShowL_nat i)) m' of None ⇒ False
          | Some (MValue val) ⇒ (case MTArray x1 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTArray x1 x12) m' (hash stl2 (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTArray x1 x12 of MTArray len' arr' ⇒ MCon (MTArray x1 x12)
m' loc2 | MTValue Types ⇒ False))"
      using MCon.simps(2)[of x11 "MTArray x1 x12" m' stl2] MTArray idef idef' by metis
    then have mdcond:"MCon (MTArray x1 x12) m' l'" using idef by simp
    then have "∀i<x1. ∃x. accessStore (hash l' (ShowL_nat i)) m' = Some x"
      by (metis bot_nat_0.not_eq_extremum less_nat_zero_code mcon_accessStore)
    have "(case accessStore (hash stl2 (ShowL_nat i)) mem' of None ⇒ False
          | Some (MValue val) ⇒ (case MTArray x1 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
              (MTArray x1 x12) mem' (hash stl2 (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTArray x1 x12 of MTArray len' arr' ⇒ MCon (MTArray x1 x12)
mem' loc2 | MTValue Types ⇒ False))"
      using MTArray.prems(5) MCon.simps(2)[of x11 "MTArray x1 x12" mem' stl2] idef idef2 idef' by metis
    then have memCond:"MCon (MTArray x1 x12) mem' l'" using idef idef2 by simp
    then have d40:"∀i<x1. ∃x. accessStore (hash l' (ShowL_nat i)) mem' = Some x" using idef2 idef
      by (metis bot_nat_0.not_eq_extremum less_nat_zero_code mcon_accessStore)
    have d90:"hash stl2 (ShowL_nat i) ≠ stl1" using MTArray.prems using LSubPrefL2_def by auto
    have "l' ≠ stl1"
    proof
      assume e1:"l' = stl1"
      then have "LSubPrefL2 l' stl1" using LSubPrefL2_def by auto
      then have "∀i. LSubPrefL2 (hash l' (ShowL_nat i)) stl1" using e1 LSubPrefL2_def by auto
      then have "∀i. accessStore (hash l' (ShowL_nat i)) mem' = None" using MTArray.prems(3) by auto
      then show False using d40
        using ‹MCon (MTArray x1 x12) mem' l'› by fastforce
    qed
    moreover have "¬TypedMemSubPrefPtrs m' x1 x12 l' stl1" using MTArray.IH[of x1 l'] using mdcond
MTArray.prems(2,3,6) memCond idef idef2
      by (metis MCon_imps_Some MTArray.IH LSubPrefL2_def hash_suffixes_associative option.discI)
    ultimately show "∃l. accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l) ∧ l ≠ stl1 ∧ ¬
TypedMemSubPrefPtrs m' x1 x12 l stl1 "
      by (simp add: idef)
  qed

  then show ?case using TypedMemSubPrefPtrs.simps(2)[of m' x11 x1 x12 stl2 stl1]
    by (metis memoryvalue.inject(2) option.inject)
next
  case (MTValue x)
  then show ?case
    by (metis LSubPrefL2_def TypedMemSubPrefPtrs.simps(1) hash_inequality)
qed

lemma cpm2mLocationRelatedToTopNotInOrig:
```

```
  assumes " LSubPrefL2 intimLoc l"
    and "¬ LSubPrefL2 stl2 l"
    and "MCon (MTArray x11 x12) m' stl2"
    and "∀ loc. LSubPrefL2 loc stl1 ⟶ accessStore loc mem' = None"
    and "MCon (MTArray x11 x12) mem' stl2"
    and "∀ locs. ¬ LSubPrefL2 locs stl1 ⟶ accessStore locs mem' = accessStore locs m'"
    and "∀ l' ≠ l. ¬(LSubPrefL2 l l')"
    and "stl1 = l"
  shows "¬TypedMemSubPrefPtrs m' x11 x12 stl2 intimLoc" using assms
proof(induction x12 arbitrary:x11 stl2)
  case (MTArray x1 x12)
  have "(∀ i<x11. ∃ l. accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l) ∧ (l ≠
      intimLoc ∧ ¬TypedMemSubPrefPtrs m' x1 x12 l intimLoc))"
  proof intros
    fix i assume d10:"i<x11"
    then obtain l' where idef:"accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l')" using
MTArray.prems
      by (metis MConArrayPointers MCon.simps(2))
    have "l ≠ l'"
    proof
      assume "l = l'"
      then have "TypedMemSubPrefPtrs m' x11 (MTArray x1 x12) stl2 l" using idef d10 by auto
      moreover have "¬ TypedMemSubPrefPtrs m' x11 (MTArray x1 x12) stl2 l"
        using cpm2mNotSubPrefsOfOriginal[of x11 "MTArray x1 x12" m' stl2 l mem'] MTArray.prems by auto
      ultimately show False by simp
    qed
    moreover have "MCon (MTArray x1 x12) mem' l'" using MTArray.prems idef d10
      by (metis CompTypeRemainsMCon Mutual_NonSub_SpecificNonSub CompMemType.simps(2))
    moreover have " MCon (MTArray x1 x12) m' l'" using MTArray.prems idef d10
      by (metis CompTypeRemainsMCon CompMemType.simps(2))
    moreover have "¬ LSubPrefL2 l' l" using calculation MTArray.prems idef LSubPrefL2_def
      by (metis MCon_imps_Some hash_suffixes_associative option.distinct(1))
    ultimately have "¬TypedMemSubPrefPtrs m' x1 x12 l' intimLoc" using MTArray.IH[of l' x1]
MTArray.prems idef by blast
    then show "∃ l. accessStore (hash stl2 (ShowL_nat i)) m' = Some (MPointer l) ∧ l ≠ intimLoc
        ∧ ¬ TypedMemSubPrefPtrs m' x1 x12 l intimLoc" using d10 idef
      using MTArray.prems(1) <¬ LSubPrefL2 l' l> by auto
  qed
  then show ?case using TypedMemSubPrefPtrs.simps(2)[of m' x11 x1 x12 stl2 intimLoc] by auto

next
  case (MTValue x)
  then have "(∀ i<x11. hash stl2 (ShowL_nat i) ≠ intimLoc)" using LSubPrefL2_def
    using Mutual_NonSub_SpecificNonSub by blast
  then show ?case using TypedMemSubPrefPtrs.simps(1)[of m' x11 x stl2 intimLoc] by blast
qed


lemma cps2mNotSubPrefsOfChanged:
  assumes "MCon (MTArray x t) m' stl1"
    and "¬ LSubPrefL2 stl2 stl1"
    and "∀ la l'. TypedStoSubpref la stl1 (STArray x' t') ∧ accessStore la m' = Some (MPointer l') ⟶
l' = la"
    and "0 < x"
    and "cps2mTypeCompatible (STArray x' t') (MTArray x t)"
  shows "¬TypedMemSubPrefPtrs m' x t stl1 stl2" using assms
proof(induction t' arbitrary:stl1 x x' t)
  case (STArray x1 t')
  then have a1:"(0 < x ∧ x' = x ∧ cps2mTypeCompatible (STArray x1 t') t)"
    using cps2mTypeCompatible.simps(2)[of x' "STArray x1 t'" x t] by blast
  then obtain t'' where tdef:"t = MTArray x1 t''" using cps2mTypeCompatible.simps by (cases t; auto)
  have d10:"∀ i<x. accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer (hash stl1 (ShowL_nat i)))"
  proof intros
    fix i
```

```
    assume idef:"i < x"
    then obtain l where ptr:"accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l)"
      using MConArrayPointers[of x x1 t'' m' stl1 i] STArray.prems(1) tdef by blast
    have sub:"TypedStoSubpref (hash stl1 (ShowL_nat i)) stl1 (STArray x' t')"
      using idef a1
      using TypedStoSubpref_sameLoc by auto
    then have "l = hash stl1 (ShowL_nat i)"
      using STArray.prems(3) ptr
      by (metis STArray.prems(3) ptr TypedStoSubpref.simps(2) idef a1)
    then show "accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer (hash stl1 (ShowL_nat i)))"
      using ptr by blast
  qed

  have "(∀i<x. ∃l. accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l) ∧ ¬(l = stl2
      ∨ hash stl1 (ShowL_nat i) = stl2 ∨ TypedMemSubPrefPtrs m' x1 t'' l stl2))"
  proof intros
    fix i
    assume idef:"i<x"
    then have ac:"accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer (hash stl1 (ShowL_nat i)))"
using d10 by simp
    have "(hash stl1 (ShowL_nat i)) ≠ stl2" using STArray LSubPrefL2_def by blast
    have mcond:"∀i<x. (case accessStore (hash stl1 (ShowL_nat i)) m' of None ⇒ False
                | Some (MValue val) ⇒ (case MTArray x1
                    t'' of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTArray x1 t'') m' (hash stl1
                    (ShowL_nat i)))
                | Some (MPointer loc2) ⇒ (case MTArray x1 t'' of MTArray len' arr' ⇒ MCon (MTArray x1
t'') m' loc2 | MTValue Types ⇒ False)) ∧
               ((∃p. accessStore stl1 m' = Some (MPointer p)) ∨ accessStore stl1 m' = None)"
      using MCon.simps(2)[of x "(MTArray x1 t'')" m' stl1] STArray.prems(1) idef tdef a1 by auto
    have " ∀la l'. TypedStoSubpref la (hash stl1 (ShowL_nat i)) (STArray x1 t') ∧ accessStore la m' =
Some (MPointer l') ⟶ l' = la"
      using STArray.prems(3) idef stoMoreSpecificTypedSubpref a1 by blast
    moreover have "MCon (MTArray x1 t'') m' (hash stl1 (ShowL_nat i))" using STArray.prems(1) ac mcond
a1 tdef
      using CompTypeRemainsMCon idef CompMemType.simps(2) by blast
    moreover have "¬ LSubPrefL2 stl2 (hash stl1 (ShowL_nat i))" using STArray.prems(2) LSubPrefL2_def
Not_Sub_More_Specific by auto
    ultimately have "¬TypedMemSubPrefPtrs m' x1 t'' (hash stl1 (ShowL_nat i)) stl2" using STArray.IH[of
x1 t''
        "(hash stl1 (ShowL_nat i))"x1] idef ac a1 tdef by fastforce
    then show "∃l. accessStore (hash stl1 (ShowL_nat i)) m' = Some (MPointer l)
            ∧ ¬ (l = stl2 ∨ hash stl1 (ShowL_nat i) = stl2 ∨ TypedMemSubPrefPtrs m' x1 t'' l stl2) "
      using ac ‹hash stl1 (ShowL_nat i) ≠ stl2› tdef by simp
  qed
  then show ?case using TypedMemSubPrefPtrs.simps(2)[of m' x x1 t'' stl1 stl2] using tdef by auto
next
  case (STMap x1 t')
  then show ?case
    by (metis cps2mTypeCompatible.simps(2,4))
next
  case (STValue x)
  then show ?case
    using LSubPrefL2_def TypedMemSubPrefPtrs.simps(1)
    by (metis cps2mTypeCompatible.simps(2) cps2mTypeCompatible.simps(6) mcon_accessStore)
qed
```

11.13 Type Compatibility Under Copying

These lemmas establish how type annotations at sub-locations are updated when copying between memory and storage regions with compatible but structurally distinct types. The key predicates are *TypedMemSubPref* and *TypedStoSubpref*, which identify which locations fall within the typed footprint of a copy destination.

Covered operations: *cpm2mrec*, *cpm2m* (memory-to-memory), *cps2mrec*, *cps2m* (storage-to-memory). The *somesome* variants additionally establish that every typed sub-location of the destination holds a value after the copy.

```
lemma cpm2mrec_array_root_both:
  assumes step: "cpm2mrec srcl destl (MTArray x t) ms md = Some v'''"
  shows "accessTypeStore destl v''' = Some (MTArray x t)
    ∧ accessStore destl v''' = Some (MPointer destl)"
proof -
  obtain l where ldef: "accessStore srcl ms = Some (MPointer l)"
    using step cpm2mrec.simps(1)[of srcl destl x t ms md]
    by (auto split: if_splits option.splits memoryvalue.splits)
  let ?m = "updateTypedStore destl (MPointer destl) (MTArray x t) md"
  have expanded:
    "Some v''' =
      (let m = updateTypedStore destl (MPointer destl) (MTArray x t) md
       in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x)"
    using step ldef by (simp add: cpm2mrec.simps(1))
  have run:
    "iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) ?m x = Some v'''"
    using expanded unfolding Let_def by simp
  have keep_type:
    "accessTypeStore destl ?m = accessTypeStore destl v'''"
    using cpm2mSingleChange_Typed[OF run] by blast
  have keep_store:
    "accessStore destl ?m = accessStore destl v'''"
    using cpm2mSingleChange[OF run] by simp
  show ?thesis
    using keep_type keep_store
    unfolding accessTypeStore_def updateTypedStore_def accessStore_def updateTypeStore_def
updateStore_def
    by simp
qed

lemma cpm2mrec_child_root_type:
  assumes step: "cpm2mrec srcl destl t ms md = Some v'''"
  shows "accessTypeStore destl v''' = Some t"
proof (cases t)
  case (MTArray x1 t')
  have step_arr: "cpm2mrec srcl destl (MTArray x1 t') ms md = Some v'''"
    using step MTArray by simp
  from cpm2mrec_array_root_both[OF step_arr]
  show ?thesis
    using MTArray by fastforce
next
  case (MTValue x)
  then obtain v where "accessStore srcl ms = Some (MValue v)"
    using step cpm2mrec.simps(2)[of srcl destl x ms md]
    by (auto split: option.splits memoryvalue.splits)
  then have "v''' = updateTypedStore destl (MValue v) (MTValue x) md"
    using step MTValue cpm2mrec.simps(2)[of srcl destl x ms md] by simp
  then show ?thesis
    using MTValue by fastforce
qed

lemma cpm2m_TypeCompChangeIndexs:
  assumes "iter' (λi m''. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m'') md x1 =
Some v'"
  shows "∀i<x1. accessTypeStore (hash destl (ShowL_nat i)) v' = Some t"
proof (rule iter'_indexed_type_writes[
    where loc = "λi. hash destl (ShowL_nat i)" and tp = t, OF assms])
  fix i j m1 m2
  assume j_less: "j < i"
    and step: "cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m1 = Some m2"
  show "accessTypeStore (hash destl (ShowL_nat j)) m2 =
      accessTypeStore (hash destl (ShowL_nat j)) m1"
    using j_less step
    by (metis ShowLNatDot cpm2mrec_SubPrefixes_both hash_int_prefix hashesIntSame nat_neq_iff)
```

```
next
  fix i m1 m2
  assume step: "cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m1 = Some m2"
  show "accessTypeStore (hash destl (ShowL_nat i)) m2 = Some t"
    using cpm2mrec_child_root_type[OF step] .
qed


lemma cpm2mrec_TypeCompChange:
  "cpm2mrec srcl destl tp ms md = Some v''' ⟶
  (∀destl'. TypedMemSubPref destl' destl tp ⟶
  (case tp of MTValue val ⇒ accessTypeStore destl v''' = Some tp
      | MTArray x' t' ⇒ (∃t''. CompMemType v''' x' t' t'' destl destl' ∧
                          (case t'' of MTArray parent_len parent_arr ⇒
                              ∀i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v''' = Some
parent_arr
                           | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
                        )))"
proof(induction tp arbitrary:srcl destl ms md v''')
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:"cpm2mrec srcl destl (MTArray x1 t) ms md = Some v''''"
      and ***:"TypedMemSubPref destl' destl (MTArray x1 t)"
    obtain l where ldef:"accessStore srcl ms = Some (MPointer l)"
      using ** cpm2mrec.simps(1)[of srcl destl x1 t ms]
      by (auto split:if_splits option.splits memoryvalue.splits)
    then have a5:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) md
                in iter' (λi m'. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m') m
x1)"
      using ** by (simp add: cpm2mrec.simps(1))
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t) md" by auto


    then have v''Def:"Some v''' = iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t
ms) m x1"
      using a5 ldef unfolding cpm2mrec.simps by presburger
    have "case MTArray x1 t of
       MTArray x' t' ⇒
         ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
               (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
                | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
      | MTValue val ⇒ accessTypeStore destl v''' = Some (MTArray x1 t)" using ***
    proof(induction rule: iter'_induct[OF _ _ v''Def[symmetric]])
      case (1 v')
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi m'. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m') m x
= Some v'"
          and a20:"(TypedMemSubPref destl' destl (MTArray x t) ⟶
          (case MTArray x t of
           MTArray x' t' ⇒
             ∃t''. CompMemType v' x' t' t'' destl destl' ∧
                   (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v' = Some parent_arr
                    | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))
           | MTValue val ⇒ accessTypeStore destl v' = Some (MTArray x t)))"
          and a30:"cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" by
blast
      then show ?case
```

```
      proof(cases "destl' = destl")
        case True
        then show ?thesis using 2
          by (metis LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality
              typedPrefix_imp_SubPref)
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
          case True
          then have "destl' ≠ hash destl (ShowL_nat x)"
            by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
          moreover have "∀ i<x. ¬ TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
t" using True
            using TypedMemSubPref_hash_index_disjoint_typed by blast
          ultimately have i1:"∀ i<x. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore
(hash destl' (ShowL_nat i)) v''"
            using cpm2mrec_SubPrefixes_typedPref_typed[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t ms v' v''] a30
            by (meson ShowLNatDot f1 hash_injective)
          obtain t'' where i2:"(CompMemType v' x t t'' destl destl' ∧
             (case t'' of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))"
            using True a20 a10 by auto


          have i5:"∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
            using a30 cpm2mrec_SubPrefixes by blast
          moreover have i6:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l (hash destl
(ShowL_nat x))"
            using TypedMemSubPref_index_disjoint by blast
          moreover have i7:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =
accessStore l v''"
            using calculation by auto
          moreover have i8:"∀ l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
            using cpm2mSelfPointers[OF a10] by blast
          show ?thesis
          proof(cases t'')
            case (MTArray x11 x12)
            then have cc0:"CompMemType v' x t (MTArray x11 x12) destl destl' ∧ (∀ i<x11.
accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
              using i2 by simp
            then have "CompMemType v'' x t (MTArray x11 x12) destl destl' ∧ (∀ i<x11. accessTypeStore
(hash destl' (ShowL_nat i)) v'' = Some x12)"
              using CompMemType_preservation_induction[OF i5 i6 i7 cc0 i8] i1
              using cpm2mrec_SubPrefixes_both a30 TypedMemSubPref_hash_not_below_index[OF True] by
blast
            then show ?thesis using CompMemType_extend_array by force
          next
            case (MTValue x2)
            then have cc0:"accessTypeStore destl' v' = Some (MTValue x2) ∧ CompMemType v' x t (MTValue
x2) destl destl'" using i2 by simp
            then have "CompMemType v'' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' =
Some (MTValue x2)"
              using CompMemTypeValue_preservation_induction[OF i5 i6 i7 cc0 i8]
              using True a30 i6 cpm2mrec_SubPrefixes_both by blast
            then show ?thesis using CompMemType_extend by fastforce
          qed
        next
          case False
```

```
        then have subs:"(∀ i<x. ¬TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∧ destl' ≠ hash
destl (ShowL_nat i))" by simp
        moreover obtain i where idef:"i<Suc x ∧ (TypedMemSubPref destl' (hash destl (ShowL_nat i)) t
∨ destl' = hash destl (ShowL_nat i))"
          using 2(3) f1 unfolding TypedMemSubPref.simps by auto
        have iIsx:"i = x"
        proof(rule ccontr)
          assume a0:"i ≠ x"
          then have "i < x ∧ (TypedMemSubPref destl' (hash destl (ShowL_nat i)) t ∨ destl' = hash
destl (ShowL_nat i))"
            using idef by simp
          then show False using subs by auto
        qed

        then have "cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" us-
ing a30 by simp

        then have IH1:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
       (case t of
         MTArray x' t' ⇒
          ∃ t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                (case t'' of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                 | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
        | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t)"
          using MTArray.IH[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat x))" ms v' v''] idef a30
iIsx by simp

        then show ?thesis
        proof(cases t)
          case (MTArray x11 x12)
          have step_arr:
            "cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTArray x11 x12) ms v' = Some
v''"
            using a30 MTArray by simp
          obtain ll where llDef:
            "accessStore (hash l (ShowL_nat x)) ms = Some (MPointer ll)"
            using step_arr cpm2mrec.simps(1)[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat x))"
x11 x12 ms v']
            by (auto split: if_splits option.splits memoryvalue.splits)
          have v''Def:
            "Some v'' = iter' (λi. cpm2mrec (hash ll (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
(ShowL_nat i)) x12 ms)
              (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 x12) v') x11"
            using step_arr llDef unfolding cpm2mrec.simps(1) Let_def by (simp)
          have root_bundle:
            "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
              ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
            using cpm2mrec_array_root_both[OF step_arr] by simp
          then show ?thesis
          proof(cases "destl' = hash destl (ShowL_nat x)")
            case True
            then have "accessTypeStore destl' v'' = Some (MTArray x11 x12)"
              using root_bundle by simp
            moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
              using MTArray True root_bundle unfolding CompMemType.simps by blast
            moreover have "∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12"
              using cpm2m_TypeCompChangeIndexs[OF v''Def[symmetric]] True by simp
            ultimately show ?thesis using MTArray True root_bundle by force
          next
            case False
            then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using subs iIsx
```

```
              using idef by blast
            then have "(case t of
     MTArray x' t' ⇒
       ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
         (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
          | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
     | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t)"
              using IH1 by simp
            then have "∃t''. CompMemType v'' x11 x12 t'' (hash destl (ShowL_nat x)) destl' ∧
           (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
              using MTArray by simp
            then show ?thesis using MTArray root_bundle by auto
          qed
        next
          case (MTValue x2)
          then have "destl' = hash destl (ShowL_nat x)" using idef iIsx by simp
          then have v''Def:"(case accessStore (hash l (ShowL_nat x)) ms of None ⇒ None
                  | Some (MValue v) ⇒ Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v)
(MTValue x2) v')
                  | Some (MPointer literal) ⇒ None) = Some v''"
            using a30 cpm2mrec.simps(2)[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 ms
v'] MTValue by simp
          then obtain v where "accessStore (hash l (ShowL_nat x)) ms = Some (MValue v)" by (auto
split:option.splits memoryvalue.splits)
          then have "v'' = (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2) v') "
using v''Def by simp
          then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
            using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def by auto
        qed
      qed

    qed
  qed
  then show "case MTArray x1 t of
  MTArray x' t' ⇒
    ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
          (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v''' = Some parent_arr
           | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
  | MTValue val ⇒ accessTypeStore destl v''' = Some (MTArray x1 t)" by simp
  qed
next
  case (MTValue x)
  show ?case
  proof intros
    fix destl'
    assume **:"cpm2mrec srcl destl (MTValue x) ms md = Some v''''"
      and ***:" TypedMemSubPref destl' destl (MTValue x)"
    then have "destl' = destl" by simp
    obtain v where "accessStore srcl ms = Some (MValue v) ∧ v''' = updateTypedStore destl (MValue v)
(MTValue x) md"
      using ** unfolding cpm2mrec.simps by (auto split:option.splits memoryvalue.splits)
    then show "(case MTValue x of
        MTArray x' t' ⇒
          ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
                (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
                 | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
        | MTValue val ⇒ accessTypeStore destl v''' = Some (MTValue x))"
      using ‹destl' = destl› unfolding updateTypedStore_def updateTypeStore_def accessTypeStore_def by
```

```
simp
  qed
qed

lemma cpm2m_TypeCompChange:
  assumes "iter' (λi m''. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m'') md x1 =
Some v'"
  shows "(∀destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶
          (∃st. CompMemType v' x1 t st destl destl' ∧
            (case st of MTArray parent_len parent_arr ⇒
              ∀i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi m''. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m'') md
x = Some v'"
      and a20: "(∀destl'.
             TypedMemSubPref destl' destl (MTArray x t) ⟶
             (∃st. CompMemType v' x t st destl destl' ∧
                   (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v' = Some parent_arr
                    | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
      and a30: "cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" by
blast
  then have "(∀destl'.
        TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
        (case t of
         MTArray x' t' ⇒
           ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                 (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                  | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
         | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t))"
    using cpm2mrec_TypeCompChange[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t ms v' v'']
by blast
  show ?case
  proof intros
    fix destl'
    assume a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"
    show "(∃st. CompMemType v'' (Suc x) t st destl destl' ∧
            (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval)))"
    proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
      case True
      then have "destl' ≠ hash destl (ShowL_nat x)" using True
        by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
      moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using True
        using TypedMemSubPref_index_disjoint_typed by blast
      ultimately have i1:"(∀destl'.
        destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v'')"
        using cpm2mrec_SubPrefixes_typedPref_typed[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t ms v' v''] a30 by simp
      obtain st where i2:" (CompMemType v' x t st destl destl' ∧
            (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))" using True a20 by
blast
```

```
      have store_preserved: "∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs
v' = accessStore locs v''"
        using a30 cpm2mrec_SubPrefixes by blast
      moreover have no_prefix_conflict: "∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l
(hash destl (ShowL_nat x))"
        using TypedMemSubPref_index_disjoint by blast
      moreover have store_consistency: "∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l
v' = accessStore l v''"
        using calculation by auto


      moreover have self_pointers:"∀ l l'. TypedMemSubPref l destl (MTArray x t)
                                  ∧ accessStore l v' = Some (MPointer l') ⟶ l' = l"
        using cpm2mSelfPointers[OF a10] by blast

      show ?thesis
      proof(cases st)
        case (MTArray x11 x12)
        have i1':"∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore (hash destl'
(ShowL_nat i)) v''"
          using i1
          using TypedMemSubPref_hash_not_below_index[OF True] TypedMemSubPref_hash_index_disjoint_typed[OF
True] LSubPrefL2_refl by metis
        then have "CompMemType v' x t (MTArray x11 x12) destl destl' ∧
                    (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
          using i2 MTArray by simp
        then have "CompMemType v'' x t (MTArray x11 x12) destl destl'
                  ∧ (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12)"
          using CompMemType_preservation_induction[OF store_preserved no_prefix_conflict
              store_consistency _self_pointers i1']
          by blast
        then show ?thesis using CompMemType_extend_array MTArray by force
      next
        case (MTValue x2)
        then have "CompMemType v' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v' = Some
(MTValue x2)"
          using i2 by auto
        then have "CompMemType v'' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' = Some
(MTValue x2)"
          using CompMemTypeValue_preservation_induction[OF store_preserved no_prefix_conflict
              store_consistency _self_pointers] using i1
          using ‹¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t› ‹destl' ≠ hash destl (ShowL_nat
x)› by blast
        then show ?thesis using CompMemType_extend by fastforce
      qed
    next
      case False
      then have "destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t"
        using a1 unfolding TypedMemSubPref.simps
        using less_SucE by blast
      then show ?thesis
      proof(cases t)
        case (MTArray x11 x12)
        have step_arr:
          "cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTArray x11 x12) ms v' = Some
v''"
          using a30 MTArray by simp
        obtain l where ldef:
          "accessStore (hash srcl (ShowL_nat x)) ms = Some (MPointer l)"
          using step_arr cpm2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11
x12 ms v']
          by (auto split: if_splits option.splits memoryvalue.splits)
        have v''Def:
```

```
          "Some v'' = iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
(ShowL_nat i)) x12 ms)
            (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x))) (MTArray
x11 x12) v') x11"
          using step_arr ldef unfolding Let_def cpm2mrec.simps(1) by (simp)
        have root_bundle:
          "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
            ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat x)))"
          using cpm2mrec_array_root_both[OF step_arr] by simp
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
          case True
          then have "accessTypeStore destl' v'' = Some (MTArray x11 x12)"
            using root_bundle by simp
          then have "∀i<x11. accessTypeStore (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) v'' = Some
x12"
            using cpm2m_TypeCompChangeIndexs[OF v''Def[symmetric]] by blast
          moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
            using MTArray True root_bundle unfolding CompMemType.simps by blast
          ultimately show ?thesis using MTArray True root_bundle mtypes.simps(5) by blast
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t"
            using ‹destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x))
t› by blast
          then have "(case t of
                      MTArray x' t' ⇒
           ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                 (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                    | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
            | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t)"
            using cpm2mrec_TypeCompChange[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t ms
v' v''] a30 by simp

          then show ?thesis using MTArray root_bundle by auto
        qed
      next
        case (MTValue x2)
        then have "destl' = hash destl (ShowL_nat x)"
          using ‹destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t›
by auto
        then obtain v where "accessStore (hash srcl (ShowL_nat x)) ms = Some (MValue v) ∧
                             (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2) v') =
v'''
          using a30 cpm2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 ms
v'] MTValue
          by (simp split:option.splits memoryvalue.splits)
        then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
          using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def by auto
      qed
    qed
  qed
qed


lemma cps2mrec_array_root_both:
  assumes step: "cps2mrec srcl destl (STArray x t) ms md = Some v'''"
    and conv: "cps2mTypeConvert t = Some tt"
  shows "accessTypeStore destl v''' = Some (MTArray x tt)
    ∧ accessStore destl v''' = Some (MPointer destl)"
proof -
  let ?m = "updateTypedStore destl (MPointer destl) (MTArray x tt) md"
```

```
  have expanded:
    "Some v''' =
      (let m = updateTypedStore destl (MPointer destl) (MTArray x tt) md
       in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x)"
    using step conv by (simp add: cps2mrec.simps(1))
  have run:
    "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) ?m x = Some v'''"
    using expanded unfolding Let_def by simp
  have keep_root:
    "accessStore destl ?m = accessStore destl v'''
      ∧ accessTypeStore destl ?m = accessTypeStore destl v''''"
    using cps2mSingleChange_both[OF run]
    accessStore_updateStore unfolding updateTypedStore_def
    by simp
  show ?thesis
    using keep_root
    unfolding updateTypedStore_def accessTypeStore_def updateTypeStore_def updateStore_def
accessStore_def
    by simp
qed

lemma cps2mrec_child_root_type:
  assumes step: "cps2mrec srcl destl t ms md = Some v''''"
    and comp: "cps2mTypeCompatible (STArray l t) (MTArray l tp)"
  shows "accessTypeStore destl v''' = Some tp"
proof (cases t)
  case (STArray x1 t')
  have step_arr: "cps2mrec srcl destl (STArray x1 t') ms md = Some v'''"
    using step STArray by simp
  obtain tt where conv: "cps2mTypeConvert t' = Some tt"
    using step_arr cps2mrec.simps(1)[of srcl destl x1 t' ms md]
    by (auto split: option.splits memoryvalue.splits)
  have tp_def: "tp = MTArray x1 tt"
  proof -
    have "cps2mTypeConvert (STArray l (STArray x1 t')) = Some (MTArray l tp)"
      using comp convertComp
      using STArray by blast
    then show ?thesis
      using STArray conv by simp
  qed
  from cps2mrec_array_root_both[OF step_arr conv]
  show ?thesis using tp_def by simp
next
  case (STMap x21 x22)
  then show ?thesis
    using step cps2mrec.simps(3) by simp
next
  case (STValue x3)
  have tp_def: "tp = MTValue x3"
  proof -
    have "cps2mTypeConvert (STArray l (STValue x3)) = Some (MTArray l tp)"
      using comp STValue convertComp
      by blast
    then show ?thesis
      using STValue by simp
  qed
  obtain v where vdef: "accessStorage x3 srcl ms = v"
    by blast
  then have "v''' = updateTypedStore destl (MValue v) (MTValue x3) md"
    using step STValue cps2mrec.simps(2)[of srcl destl x3 ms md] by simp
  then show ?thesis
    using tp_def by simp
qed
```

```
lemma cps2m_TypeCompChangeIndexs:
  assumes "iter' (λi m''. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m'') md x1 =
Some v'"
    and "cps2mTypeCompatible (STArray x1 t) (MTArray x1 t')"
  shows "∀ i<x1. accessTypeStore (hash destl (ShowL_nat i)) v' = Some t'"
proof (rule iter'_indexed_type_writes[
    where loc = "λi. hash destl (ShowL_nat i)" and tp = t', OF assms(1)])
  fix i j m1 m2
  assume j_less: "j < i"
    and step: "cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m1 = Some m2"
  show "accessTypeStore (hash destl (ShowL_nat j)) m2 =
      accessTypeStore (hash destl (ShowL_nat j)) m1"
    using j_less step
    by (metis ShowLNatDot cps2mrec_SubPrefixes_both hash_int_prefix hashesIntSame nat_neq_iff)
next
  fix i m1 m2
  assume step: "cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m1 = Some m2"
  show "accessTypeStore (hash destl (ShowL_nat i)) m2 = Some t'"
    using cps2mrec_child_root_type[OF step assms(2)] .
qed

lemma cps2mrec_TypeCompChange:
  "cps2mrec srcl destl t' ms md = Some v''' ⟶ cps2mTypeCompatible (STArray l t') (MTArray l tp)⟶
  (∀ destl'. TypedMemSubPref destl' destl tp ⟶
  (case tp of MTValue val ⇒ accessTypeStore destl v''' = Some tp
      | MTArray x' t' ⇒ (∃ t''. CompMemType v''' x' t' t'' destl destl' ∧
                          (case t'' of MTArray parent_len parent_arr ⇒
                              ∀ i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) v''' = Some
parent_arr
                           | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
                        )))"
proof(induction t' arbitrary:srcl destl ms md v''' l tp)
  case (STArray x1 t')
  show ?case
  proof intros
    fix destl'
    assume **:"cps2mrec srcl destl (STArray x1 t') ms md = Some v''''"
      and ****:"cps2mTypeCompatible (STArray l (STArray x1 t')) (MTArray l tp)"
      and ***:"TypedMemSubPref destl' destl tp"
    obtain t'' where ldef:"cps2mTypeConvert t' = Some t''"
      using ** cps2mrec.simps(1)[of srcl destl x1 t' ms] by (auto split:if_splits option.splits
memoryvalue.splits)
    then have a5:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t'') md
                               in iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i))
t' ms) m x1)"
      using ** by (simp add: cps2mrec.simps(1))
    then obtain m where mdef:"m = updateTypedStore destl (MPointer destl) (MTArray x1 t'') md " by
auto
    then have v''Def:"Some v''' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat
i)) t' ms) m x1"
      using a5 ldef unfolding cpm2mrec.simps by presburger
    obtain t''' where t'''def:"tp = MTArray x1 t'''" using ****
      by (metis mtypes.exhaust cps2mTypeCompatible.simps(2) cps2mTypeCompatible.simps(3))
    then have ****:"cps2mTypeCompatible (STArray x1 t') (MTArray x1 t''')" using **** by simp
    have ***:"TypedMemSubPref destl' destl (MTArray x1 t''')" using t'''def *** by blast
    have "∃ t''. CompMemType v''' x1 t''' t'' destl destl' ∧
              (case t'' of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
               | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))" using *** ****
    proof(induction rule: iter'_induct[OF _ _ v''Def[symmetric]])
      case (1 v')
      then show ?case
        by (simp add: t'''def)
    next
```

```
      case (2 x v'')
      then obtain v'
        where a10:" iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' ms) m x =
Some v'"
          and a20:"(TypedMemSubPref destl' destl (MTArray x t''')) ⟶
          cps2mTypeCompatible (STArray x t') (MTArray x t''') ⟶
          (∃t''. CompMemType v' x t''' t'' destl destl' ∧
                 (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v' = Some parent_arr
                  | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))"
          and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' ms v' = Some v''" by
blast
      then show ?case
      proof(cases "destl' = destl")
        case True
        then show ?thesis using 2
          by (metis LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality
              typedPrefix_imp_SubPref)
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref destl' destl (MTArray x t''')")
          case True
          then have "destl' ≠ hash destl (ShowL_nat x)"
            by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
          moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t''''" using True
            using TypedMemSubPref_index_disjoint_typed by blast
          then have "¬ TypedStoSubpref destl' (hash destl (ShowL_nat x)) t'"
            using "2.prems"(2) compatible_TypedStoSubpref_imps_TypedMemSubPref
            by (simp add: calculation)
          ultimately have i1:"accessTypeStore destl' v' = accessTypeStore destl' v''"
            using cps2mrec_SubPrefixes_typedPref_both[of "(hash srcl (ShowL_nat x))" "(hash destl
(ShowL_nat x))" t' ms v' v''] a30 by blast

          obtain t'' where i2:"(CompMemType v' x t''' t'' destl destl' ∧
             (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))"
            using True a20 a10 2 by auto


          have i5:"∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
            using a30 cps2mrec_SubPrefixes_both by blast
          moreover have i6:"∀l. TypedMemSubPref l destl (MTArray x t''') ⟶ ¬ LSubPrefL2 l (hash
destl (ShowL_nat x))"
            using TypedMemSubPref_index_disjoint by blast
          moreover have i7:"∀l. TypedMemSubPref l destl (MTArray x t''') ⟶ accessStore l v' =
accessStore l v''"
            using calculation by auto
          moreover have i8:"∀l l'. TypedMemSubPref l destl (MTArray x t''') ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
          proof (intro allI impI)
            fix l l'
            assume asm:"TypedMemSubPref l destl (MTArray x t''') ∧ accessStore l v' = Some (MPointer
l')"
            have l_neq:"l ≠ destl"
            proof
              assume "l = destl"
              then have "TypedMemSubPref destl destl (MTArray x t''')" using asm by blast
              then obtain i where "i < x ∧ (TypedMemSubPref destl (hash destl (ShowL_nat i)) t''' ∨
destl = hash destl (ShowL_nat i))"
                unfolding TypedMemSubPref.simps by blast
```

```
            then show False using hash_inequality
              by (metis hash_inequality LSubPrefL2_def hash_flatten_right neg_MemLSubPrefL2_imps_TypedMemSubPref)
          qed
          have sto_sub_root:"TypedStoSubpref l destl (STArray x t')"
            using "2.prems"(2) asm compatible_TypedStoSubpref_imps_TypedMemSubPref l_neq
            by auto
          then show "l' = l"
            using cps2mSelfPointers[OF a10] asm l_neq by blast
        qed
        show ?thesis
        proof(cases t'')
          case (MTArray x11 x12)
          then have cc0:"CompMemType v' x t''' (MTArray x11 x12) destl destl' ∧
                              (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
            using i2 by simp
          then have "CompMemType v'' x t''' (MTArray x11 x12) destl destl' ∧
                        (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12)"
            using CompMemType_preservation_induction[OF i5 i6 i7 cc0 i8] i1
            using cps2mrec_SubPrefixes_both a30 TypedMemSubPref_hash_not_below_index[OF True] by
blast
          then show ?thesis using CompMemType_extend_array by force
        next
          case (MTValue x2)
          then have cc0:"accessTypeStore destl' v' = Some (MTValue x2) ∧ CompMemType v' x t'''
(MTValue x2) destl destl'" using i2 by simp
          then have "CompMemType v'' x t''' (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' =
Some (MTValue x2)"
            using CompMemTypeValue_preservation_induction[OF i5 i6 i7 cc0 i8]
            using True a30 i6 cps2mrec_SubPrefixes_both by blast
          then show ?thesis using CompMemType_extend by fastforce
        qed

      next
        case False
        then have subs:"(∀ i<x. ¬TypedMemSubPref destl' (hash destl (ShowL_nat i)) t''' ∧ destl' ≠
hash destl (ShowL_nat i))" by simp
        moreover obtain i where idef:"i<Suc x ∧ (TypedMemSubPref destl' (hash destl (ShowL_nat i))
t''' ∨ destl' = hash destl (ShowL_nat i))"
          using 2(3) f1 unfolding TypedMemSubPref.simps by auto
        have iIsx:"i = x"
        proof(rule ccontr)
          assume a0:"i ≠ x"
          then have "i < x ∧ (TypedMemSubPref destl' (hash destl (ShowL_nat i)) t''' ∨ destl' = hash
destl (ShowL_nat i))"
            using idef by simp
          then show False using subs by auto
        qed

        then have "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' ms v' = Some v''"
using a30 by simp

        then have IH1:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t''' ⟶
       (case t''' of
        MTArray x' t' ⇒
          ∃ t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                (case t'' of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                 | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
         | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t''')"
          using STArray.IH[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" ms v' v''] idef
a30 iIsx
          using "2.prems"(2) by presburger

        then show ?thesis
```

```
          proof(cases t')
            case (STArray x11 x12)
            then obtain tF where tfDef: "t''' = MTArray x11 tF"
              by (metis "2.prems"(2) mtypes.exhaust cps2mTypeCompatible.simps(2,3))
            then have a:"Some v'' = (case cps2mTypeConvert x12 of None ⇒ None
                                | Some t' ⇒
                                let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 t') v'
                                in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x))
                                    (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
                                    (ShowL_nat i)) x12 ms) m x11)"
              using a30 cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11
x12 ms v'] STArray by auto
            then obtain ll where llDef:"cps2mTypeConvert x12 = Some ll" by (auto split:option.splits
memoryvalue.splits)

            then have a':"let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 ll) v'
                                in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x))
                                    (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
                                    (ShowL_nat i)) x12 ms) m x11 = Some v''" using a by simp
            have step_arr:
              "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STArray x11 x12) ms v' =
Some v''"
              using a30 STArray by simp
            have v''Def:
              "Some v'' = iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat
                  i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12 ms)
                (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 ll) v') x11"
              using a' llDef unfolding Let_def by simp
            have root_bundle:
              "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 ll)
                ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
              using cps2mrec_array_root_both[OF step_arr llDef] by simp
            then show ?thesis
            proof(cases "destl' = hash destl (ShowL_nat x)")
              case True
              then have "accessTypeStore destl' v'' = Some (MTArray x11 ll)"
                using root_bundle by simp
              moreover have "CompMemType v'' (Suc x) (MTArray x11 ll) (MTArray x11 ll) destl destl'"
                using True root_bundle unfolding CompMemType.simps by blast
              moreover have "∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some ll"
                using cps2m_TypeCompChangeIndexs[OF v''Def[symmetric]] True 2(4) llDef
                using STArray convertComp tfDef by auto
              moreover have "t''' = MTArray x11 ll"
              proof -
                have "cps2mTypeCompatible x12 tF" using "2.prems"(2) STArray tfDef by simp
                then show ?thesis using convertComp llDef tfDef by simp
              qed
              ultimately show ?thesis by (auto simp del: CompMemType.simps intro!: exI[of _ "MTArray
x11 ll"])
            next
              case False
              then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t'''" using subs iIsx
                using idef by blast
              then have "(case t''' of
      MTArray x' t' ⇒
        ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
              (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) v'' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
      | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t''')"
```

```
            using IH1 by simp
          then show ?thesis using STArray root_bundle t'''def 2(4)
            using tfDef by auto
        qed
      next
        case (STMap x21 x22)
        then show ?thesis using 2 by simp
      next
        case (STValue x3)
        then have destl'Def:"destl' = hash destl (ShowL_nat x)" using idef iIsx 2(4)
          using convertComp by force
        then have v''Def:"Some v'' = (let v = accessStorage x3 (hash srcl (ShowL_nat x)) ms in Some
            (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x3) v'))"
          using a30 cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x3
ms v'] STValue by simp
        then obtain v where "accessStorage x3 (hash srcl (ShowL_nat x)) ms = v" by (auto
split:option.splits memoryvalue.splits)
        then have "v'' = (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x3) v') "
using v''Def by simp
        then have "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTValue x3)"
          unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def updateStore_def by
simp
        then show ?thesis using STValue destl'Def convertComp 2(4) by fastforce
      qed
    qed
  qed
  then show "case tp of
     MTArray x' t' ⇒
       ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
             (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
     | MTValue val ⇒ accessTypeStore destl v''' = Some tp"
    by (simp add: t'''def)
qed
next
  case (STMap x1 t')
  then show ?case by simp
next
  case (STValue x)
  show ?case
  proof intros
    fix destl'
    assume **:"cps2mrec srcl destl (STValue x) ms md = Some v''''"
      and ****:"cps2mTypeCompatible (STArray l (STValue x)) (MTArray l tp)"
      and ***:"TypedMemSubPref destl' destl tp"
    then have "destl' = destl"
      by (meson TypedStoSubpref.simps(1) compatible_TypedStoSubpref_imps_TypedMemSubPref
cps2mTypeCompatible.simps(2))
    obtain v where "accessStorage x srcl ms = v ∧ Some (updateTypedStore destl (MValue v) (MTValue x)
md) = Some v''''"
      using ** unfolding cps2mrec.simps by (auto split:option.splits memoryvalue.splits)
    then show "case tp of
       MTArray x' t' ⇒
         ∃t''. CompMemType v''' x' t' t'' destl destl' ∧
               (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v''' = Some parent_arr
                | MTValue pval ⇒ accessTypeStore destl' v''' = Some (MTValue pval))
       | MTValue val ⇒ accessTypeStore destl v''' = Some tp"
      using ‹destl' = destl› **** convertComp unfolding updateTypedStore_def updateTypeStore_def
accessTypeStore_def
      by fastforce
  qed
```

qed

lemma *cps2m_TypeCompChange:*
assumes *"iter' (λi m''. cps2mrec (hash srcl (ShowL$_{nat}$ i)) (hash destl (ShowL$_{nat}$ i)) t' ms m'') md x1 = Some v'"*
and *"cps2mTypeCompatible (STArray x1 t') (MTArray x1 t)"*
shows *"(∀destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶*
(∃st. CompMemType v' x1 t st destl destl' ∧
(case st of MTArray parent_len parent_arr ⇒
∀i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v' = Some parent_arr
| MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval))))" **using** *assms(2)*
proof(*induction rule: iter'_induct[OF _ _ assms(1)]*)
case *(1)*
then show *?case* **by** *simp*
next
case *(2 x v'')*
then obtain *v'*
where *a10: " iter' (λi. cps2mrec (hash srcl (ShowL$_{nat}$ i)) (hash destl (ShowL$_{nat}$ i)) t' ms) md x = Some v'"*
and *a20: "(cps2mTypeCompatible (STArray x t') (MTArray x t) ⟶*
(∀destl'.
TypedMemSubPref destl' destl (MTArray x t) ⟶
(∃st. CompMemType v' x t st destl destl' ∧
(case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v' = Some parent_arr
| MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))))"
and *a30: "cps2mrec (hash srcl (ShowL$_{nat}$ x)) (hash destl (ShowL$_{nat}$ x)) t' ms v' = Some v''"* **by** *blast*
then have *"(∀destl'.*
TypedMemSubPref destl' (hash destl (ShowL$_{nat}$ x)) t ⟶
(case t of
MTArray x' t' ⇒
∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL$_{nat}$ x)) destl' ∧
(case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v'' = Some parent_arr
| MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
| MTValue val ⇒ accessTypeStore (hash destl (ShowL$_{nat}$ x)) v'' = Some t))"
using *"2.prems" cps2mrec_TypeCompChange* **by** *presburger*
show *?case*
proof *intros*
fix *destl'*
assume *a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"*
show *"∃st. CompMemType v'' (Suc x) t st destl destl' ∧*
(case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v'' = Some parent_arr
| MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
proof(*cases "TypedMemSubPref destl' destl (MTArray x t)"*)
case *True*
then have *"destl' ≠ hash destl (ShowL$_{nat}$ x)"* **using** *True*
by *(metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame not_less_iff_gr_or_eq typedPrefix_imp_SubPref)*
moreover have *"¬ TypedMemSubPref destl' (hash destl (ShowL$_{nat}$ x)) t"* **using** *True*
using *TypedMemSubPref_index_disjoint_typed* **by** *blast*
then have *"¬ TypedStoSubpref destl' (hash destl (ShowL$_{nat}$ x)) t'"*
using *"2.prems" compatible_TypedStoSubpref_imps_TypedMemSubPref*
by *(simp add: calculation)*
ultimately have *i1:"accessTypeStore destl' v' = accessTypeStore destl' v''"*
using *cps2mrec_SubPrefixes_typedPref_both[of "(hash srcl (ShowL$_{nat}$ x))" "(hash destl (ShowL$_{nat}$ x))" t' ms v' v''] a30* **by** *blast*

obtain *t''* **where** *i2:"(CompMemType v' x t t'' destl destl' ∧*
(case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) v' = Some parent_arr

```
      | MTValue pval ⇒ accessTypeStore destl' v' = Some (MTValue pval)))"
      using True a20 a10 2 by auto


    have i5:"∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
      using a30 cps2mrec_SubPrefixes_both by blast
    moreover have i6:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l (hash destl
(ShowL_nat x))"
      using TypedMemSubPref_index_disjoint by blast
    moreover have i7:"∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' = accessStore
l v''"
      using calculation by auto
    moreover have i8:"∀ l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some
(MPointer l') ⟶ l' = l"
    proof (intro allI impI)
      fix l l'
      assume asm:"TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer l')"
      have l_neq:"l ≠ destl"
      proof
        assume "l = destl"
        then have "TypedMemSubPref destl destl (MTArray x t)" using asm by blast
        then obtain i where "i < x ∧ (TypedMemSubPref destl (hash destl (ShowL_nat i)) t ∨ destl =
hash destl (ShowL_nat i))"
          unfolding TypedMemSubPref.simps by blast
        then show False using hash_inequality
          by (metis hash_inequality LSubPrefL2_def hash_flatten_right neg_MemLSubPrefL2_imps_TypedMemSubPref)
      qed
      have sto_sub_root:"TypedStoSubpref l destl (STArray x t')"
        using "2.prems" asm compatible_TypedStoSubpref_imps_TypedMemSubPref l_neq by auto
      then show "l' = l"
        using cps2mSelfPointers[OF a10] asm l_neq by blast
    qed
    show ?thesis
    proof(cases t'')
      case (MTArray x11 x12)
      then have cc0:"CompMemType v' x t (MTArray x11 x12) destl destl' ∧
                     (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v' = Some x12)"
        using i2 by simp
      then have "CompMemType v'' x t (MTArray x11 x12) destl destl' ∧
                 (∀ i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12)"
        using CompMemType_preservation_induction[OF i5 i6 i7 cc0 i8] i1
        using cps2mrec_SubPrefixes_both a30 TypedMemSubPref_hash_not_below_index[OF True] by blast
      then show ?thesis using CompMemType_extend_array by force
    next
      case (MTValue x2)
      then have cc0:"accessTypeStore destl' v' = Some (MTValue x2) ∧ CompMemType v' x t (MTValue x2)
destl destl'" using i2 by simp
      then have "CompMemType v'' x t (MTValue x2) destl destl' ∧ accessTypeStore destl' v'' = Some
(MTValue x2)"
        using CompMemTypeValue_preservation_induction[OF i5 i6 i7 cc0 i8]
        using True a30 i6 cps2mrec_SubPrefixes_both by blast
      then show ?thesis using CompMemType_extend by fastforce
    qed
  next
    case False
    then have "destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t"
      using a1 unfolding TypedMemSubPref.simps
      using less_SucE by blast
    then show ?thesis
    proof(cases t)
      case (MTArray x11 x12)
      then obtain x12' where t'Def:"t' = STArray x11 x12'" using 2(3)
        by (metis cps2mTypeCompatible.elims(2) cps2mTypeCompatible.simps(2,6))
```

```
        then have a:"Some v'' = (case cps2mTypeConvert x12' of None ⇒ None
     | Some t' ⇒
         let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 t') v'
         in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i)) (hash (hash
             destl (ShowL_nat x)) (ShowL_nat i)) x12' ms) m x11)"
          using a30 cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11 x12'
ms v'] by auto
        then have x12'Def:"cps2mTypeConvert x12' = Some x12" using convertComp MTArray t'Def
          using "2.prems" by force
        have step_arr:
          "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STArray x11 x12') ms v' = Some
v''"
          using a30 MTArray t'Def by simp
        have v''Def:
          "Some v'' = iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i))
              (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12' ms)
            (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x))) (MTArray
x11 x12) v') x11"
          using a x12'Def unfolding Let_def by simp
        have root_bundle:
          "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 x12)
            ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat x)))"
          using cps2mrec_array_root_both[OF step_arr x12'Def] by simp
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
          case True
          then have "accessTypeStore destl' v'' = Some (MTArray x11 x12)"
            using root_bundle by simp
          moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
            using MTArray True root_bundle unfolding CompMemType.simps by blast
          moreover have "∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12"
            using cps2m_TypeCompChangeIndexs[OF v''Def[symmetric]] True
            using MTArray assms(2) cps2mTypeCompatible.simps(2) t'Def by blast
          ultimately show ?thesis using MTArray True root_bundle by force
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t"
            using ‹destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x))
t› by blast
          then have "(case t of
                      MTArray x' t' ⇒
                       ∃t''. CompMemType v'' x' t' t'' (hash destl (ShowL_nat x)) destl' ∧
                    (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                     | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))
                        | MTValue val ⇒ accessTypeStore (hash destl (ShowL_nat x)) v'' = Some t)"
            using cps2mrec_TypeCompChange[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t'
ms v' v''] 2(3) a30
            by auto
          then have "∃t''. CompMemType v'' x11 x12 t'' (hash destl (ShowL_nat x)) destl' ∧
                (case t'' of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash
destl' (ShowL_nat i)) v'' = Some parent_arr
                                | MTValue pval ⇒ accessTypeStore destl' v'' = Some (MTValue pval))"
            using MTArray by auto
          then show ?thesis using MTArray root_bundle by auto
        qed
      next
        case (MTValue x2)
        then have "destl' = hash destl (ShowL_nat x)"
          using ‹destl' = hash destl (ShowL_nat x) ∨ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t›
by auto
        then have "Some v'' = (let v = accessStorage x2 (hash srcl (ShowL_nat x)) ms in Some
(updateTypedStore (hash
```

```
          destl (ShowL_nat x)) (MValue v) (MTValue x2) v'))"
        using a30 cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 ms
v'] MTValue
        by (metis "2.prems" stypes.exhaust cps2mTypeCompatible.simps(1,2,3,4))
      then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
        using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def by auto
    qed
  qed
 qed
qed

lemma cps2mrec_somesome:
 "cps2mrec srcl destl tp ms a = Some v''' ⟶ cps2mTypeCompatible tp tp' ⟶
 (∀destl'. TypedMemSubPref destl' destl tp' ⟶ ((∃t. accessStore destl' v''' = Some t)
    ⟷ (∃tt. accessTypeStore destl' v''' = Some tt)))"
proof(induction tp arbitrary:destl a v''' srcl tp')
 case (STArray x1 t)
 show ?case
 proof intros
  fix destl'
  assume **:"cps2mrec srcl destl (STArray x1 t) ms a = Some v'''"
   and ****:"cps2mTypeCompatible (STArray x1 t) tp'"
   and ***:"TypedMemSubPref destl' destl tp'"
  have a5:"Some v'''
        = (case cps2mTypeConvert t of None ⇒ None
   | Some t' ⇒ let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a in iter' (λi.
cps2mrec (hash srcl
      (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x1)"
   using ** cps2mrec.simps(1)[of srcl destl x1 t ms a] by simp
  then obtain t' where ldef:"cps2mTypeConvert t = Some t'" by (auto split:option.splits
memoryvalue.splits)
  then have a6:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t') a in
iter' (λi. cps2mrec (hash srcl
     (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x1)"
   using a5 by auto
  then obtain m where mdef:"updateTypedStore destl (MPointer destl) (MTArray x1 t') a = m" by auto
  then have v''def:"Some v''' = iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat
i)) t ms) m x1" using a6 by simp
  have tp'Def:"tp' = MTArray x1 t'" using **** ldef
   using convertComp by fastforce
  then have ****:"cps2mTypeCompatible (STArray x1 t) (MTArray x1 t')" using **** by blast
  have ***:"TypedMemSubPref destl' destl (MTArray x1 t')" using *** tp'Def by blast
  have "(∃t. accessStore destl' v''' = Some t) = (∃tt. accessTypeStore destl' v''' = Some tt)"
   using *** ****
  proof(induction rule: iter'_induct[OF _ _ v''def[symmetric]])
   case (1)
   then show ?case by simp
  next
   case (2 x v'')
   then obtain v'
    where a10:"iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x =
Some v'"
     and a20:"(TypedMemSubPref destl' destl (MTArray x t') ⟶
     cps2mTypeCompatible (STArray x t) (MTArray x t') ⟶ (∃t. accessStore destl' v' = Some t) =
(∃tt. accessTypeStore destl' v' = Some tt))"
     and a30:"cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" by
blast
   then show ?case
   proof(cases "destl' = destl")
    case True
    then show ?thesis using a10 a30
     by (metis "***" LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality
typedPrefix_imp_SubPref)
```

```
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref destl' destl (MTArray x t')")
          case True
          then have "destl' ≠ hash destl (ShowL_nat x)"
            by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
          moreover have "∀ i<x. ¬ TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
t'" using True
            using TypedMemSubPref_hash_index_disjoint_typed by blast
          moreover have "(∀ destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedStoSubpref destl'
              (hash destl (ShowL_nat x)) t ⟶ accessTypeStore destl' v' = accessTypeStore destl' v'')"
            using cps2mrec_SubPrefixes_typedPref_both[of "(hash srcl (ShowL_nat x))" "(hash destl
(ShowL_nat x))" t ms v' v'' ] a30 by blast
          moreover have "(∀ destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl' (hash
destl (ShowL_nat x)) t'
                              ⟶ accessTypeStore destl' v' = accessTypeStore destl' v'')"
            using 2(4) compatible_TypedStoSubpref_imps_TypedMemSubPref calculation by simp
          ultimately have i1:"∀ i<x. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore
(hash destl' (ShowL_nat i)) v''"
            by (meson ShowLNatDot f1 hash_injective)
          have i2:"(∃ t. accessStore destl' v' = Some t) = (∃ tt. accessTypeStore destl' v' = Some tt)"
            using True a20
            using "***" "2.prems"(2) by force

          have i5:"∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
            using a30 cps2mrec_SubPrefixes_both by blast
          moreover have i6:"∀ l. TypedMemSubPref l destl (MTArray x t') ⟶ ¬ LSubPrefL2 l (hash destl
(ShowL_nat x))"
            using TypedMemSubPref_index_disjoint by blast
          ultimately have i7:"∀ l. TypedMemSubPref l destl (MTArray x t') ⟶ accessStore l v' =
accessStore l v''" by auto
          have i8:"∀ l l'. TypedMemSubPref l destl (MTArray x t') ∧ accessStore l v' = Some (MPointer
l') ⟶ l' = l"
          proof (intro allI impI)
            fix l l'
            assume asm:"TypedMemSubPref l destl (MTArray x t') ∧ accessStore l v' = Some (MPointer
l')"
            have l_neq:"l ≠ destl"
            proof
              assume "l = destl"
              then have "TypedMemSubPref destl destl (MTArray x t')" using asm by blast
              then obtain i where "i < x ∧ (TypedMemSubPref destl (hash destl (ShowL_nat i)) t' ∨
destl = hash destl (ShowL_nat i))"
                unfolding TypedMemSubPref.simps by blast
              then show False using hash_inequality
                by (metis hash_inequality LSubPrefL2_def hash_flatten_right neg_MemLSubPrefL2_imps_TypedMemSubPref)

            qed
            have sto_sub_root:"TypedStoSubpref l destl (STArray x t)"
              using 2(4) asm compatible_TypedStoSubpref_imps_TypedMemSubPref l_neq by auto
            then show "l' = l"
              using cps2mSelfPointers[OF a10] asm l_neq by blast
          qed

          show ?thesis using i2
            using True a30 i6 i7 cps2mrec_SubPrefixes_both by auto
        next
          case False
          then have b5:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t' ∨ destl' = hash destl
(ShowL_nat x)"
            using 2 TypedMemSubPref.simps(2)[of destl' destl "(Suc x)" t'] f1
```

```
            using less_Suc_eq
            by (metis TypedMemSubPref.simps(2))
          have "¬ TypedStoSubpref destl' destl (STArray x t)"
            using 2(4) compatible_TypedStoSubpref_imps_TypedMemSubPref
            using False
            by (simp add: f1)
          then have "accessTypeStore destl' m = accessTypeStore destl' v''"
            using cps2mSingleChange_typedPref_both[of srcl destl t ms m x v'] a10 f1 False by blast
          then have "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''"
using a30 by simp
          then have IH1:"(∀ destl'. TypedMemSubPref destl' (hash destl (ShowL_nat x)) t' ⟶
                              (∃ t. accessStore destl' v'' = Some t) = (∃ tt. accessTypeStore destl' v'' =
Some tt))"
            using 2(4) STArray.IH[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" v' v'' t']
by simp

          have "(∃ t. accessStore destl' v'' = Some t) = (∃ tt. accessTypeStore destl' v'' = Some tt)"
          proof(cases "t")
            case (STArray x11 x12)
            then have a:"Some v'' = (case cps2mTypeConvert x12 of None ⇒ None
                                | Some t' ⇒
                              let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 t') v'
                              in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat
                                  i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12 ms) m x11)"
              using a30 cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11
x12 ms v']
              by simp
            then obtain tpp where lldef:"cps2mTypeConvert x12 = Some tpp" by (auto
split:option.splits memoryvalue.splits)
            have step_arr:
              "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STArray x11 x12) ms v' =
Some v''"
              using a30 STArray by simp
            have v''Def:
              "Some v'' = iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat
                  i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12 ms)
                (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 tpp) v') x11"
              using lldef unfolding Let_def
              by (metis (lifting) Option.option.simps(5) a)
            have root_bundle:
              "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 tpp)
                ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
              using cps2mrec_array_root_both[OF step_arr lldef] by simp
            then show ?thesis
              using IH1 root_bundle b5 by auto
          next
            case (STMap x21 x22)
            then show ?thesis using 2 by simp
          next
            case (STValue x2)
            then show ?thesis using IH1 b5
              using ldef by auto
          qed
          then show ?thesis by blast
        qed
      qed
    qed
    then show "(∃ t. accessStore destl' v''' = Some t) = (∃ tt. accessTypeStore destl' v''' = Some tt)"
by auto
  qed
next
```

```
  case (STMap x1 tp)
  then show ?case by simp
next
  case (STValue x)
  show ?case
  proof intros
    fix destl'
    assume **:"cps2mrec srcl destl (STValue x) ms a = Some v'''"
      and ****:"cps2mTypeCompatible (STValue x) tp'"
    assume ***:"TypedMemSubPref destl' destl tp'"

    have tp'Def:"tp' = (MTValue x)" using **** cps2mTypeCompatible.simps
      using convertComp by force
    then have mdef:"Some v''' = (let v = accessStorage x srcl ms in Some (updateTypedStore destl
(MValue v) (MTValue x) a))"
      using ** cps2mrec.simps(2)[of srcl destl x ms a] by simp
    then obtain v where vdef: "accessStorage x srcl ms = v" by (auto split:option.splits
memoryvalue.splits)
    then have v'''def:"Some v''' = Some (updateTypedStore destl (MValue v) (MTValue x) a)" using mdef
by simp
    then have "accessTypeStore destl v''' = Some (MTValue x)" using mdef
      unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
    moreover have "accessStore destl' v''' = Some (MValue v)"
      using v'''def unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def
      using "***" tp'Def by simp
    ultimately show "(∃ t. accessStore destl' v''' = Some t) = (∃ tt. accessTypeStore destl' v''' = Some
tt)"
      using "***" tp'Def by fastforce
  qed
qed

lemma cps2m_TypeCompChange_somesome:
  assumes "iter' (λi m''. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' ms m'') md x1
= Some v'"
    and "cps2mTypeCompatible (STArray x1 t') (MTArray x1 t)"
  shows "(∀ destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶
         ((∃ t. accessStore destl' v' = Some t) ⟷ (∃ tt. accessTypeStore destl' v' = Some tt)))"
using assms(2)
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cps2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t' ms) md x =
Some v'"
      and a20: "(cps2mTypeCompatible (STArray x t') (MTArray x t) ⟶
          (∀ destl'. TypedMemSubPref destl' destl (MTArray x t) ⟶ (∃ t. accessStore destl' v' = Some t)
= (∃ tt. accessTypeStore destl' v' = Some tt)))"
      and a30: " cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t' ms v' = Some v''" by
blast

  show ?case
  proof intros
    fix destl'
    assume a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"
    show "((∃ t. accessStore destl' v'' = Some t) ⟷ (∃ tt. accessTypeStore destl' v'' = Some tt))"
    proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
      case True
      then have "destl' ≠ hash destl (ShowL_nat x)" using True
        by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
```

```
      moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using True
        using TypedMemSubPref_index_disjoint_typed by blast
      ultimately have i1:"∀destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedStoSubpref destl'
          (hash destl (ShowL_nat x)) t' ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v''"
        using cps2mrec_SubPrefixes_typedPref_both[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat
x))" t' ms v' v''] a30 by blast
      then have i1':"∀destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl'
          (hash destl (ShowL_nat x)) t ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v''" using 2(3)
        by (simp add: compatible_TypedStoSubpref_imps_TypedMemSubPref)
      then have i2:"(∃t. accessStore destl' v' = Some t) = (∃tt. accessTypeStore destl' v' = Some tt)"
        using i1 True a20 2(3) by auto

      have store_preserved: "∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs
v' = accessStore locs v''"
        using a30 cps2mrec_SubPrefixes_both by blast
      moreover have no_prefix_conflict: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l
(hash destl (ShowL_nat x))"
        using TypedMemSubPref_index_disjoint by blast
      ultimately have store_consistency: "∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l
v' = accessStore l v''"
        by auto

      show ?thesis using i2
        using True ‹¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t› ‹destl' ≠ hash
            destl (ShowL_nat x)› i1 store_consistency
        by (simp add: i1')

    next
      case False
      then have a2:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl (ShowL_nat
x)"
        using a1 unfolding TypedMemSubPref.simps
        using less_SucE by blast
      then show ?thesis
      proof(cases t')
        case (STArray x11 x12)
        then have a:"Some v'' = (case cps2mTypeConvert x12 of None ⇒ None
     | Some t' ⇒
         let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 t') v'
         in iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i)) (hash (hash
             destl (ShowL_nat x)) (ShowL_nat i)) x12 ms) m x11)"
          using a30 cps2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11 x12
ms v'] by auto
        then obtain l where ldef:"cps2mTypeConvert x12 = Some l" by (auto split:option.splits
memoryvalue.splits)
        have step_arr:
          "cps2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) (STArray x11 x12) ms v' = Some
v''"
          using a30 STArray by simp
        have v''Def:
          "Some v'' = iter' (λi. cps2mrec (hash (hash srcl (ShowL_nat x)) (ShowL_nat i))
              (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12 ms)
            (updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x))) (MTArray
x11 l) v') x11"
          by (metis (lifting) Option.option.simps(5) a ldef)
        have root_bundle:
          "accessTypeStore (hash destl (ShowL_nat x)) v'' = Some (MTArray x11 l)
            ∧ accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat x)))"
          using cps2mrec_array_root_both[OF step_arr ldef] by simp
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
```

```
          case True
          then have "∀i<x11. TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
(MTArray x11 l)" by auto
          then have "∀i<x11. accessTypeStore (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) v'' = Some
l"
            using cps2m_TypeCompChangeIndexs[OF v''Def[symmetric]] 2(3) ldef
            by (metis mtypes.exhaust STArray convertComp cps2mTypeCompatible.simps(2,3))
          moreover have "CompMemType v'' (Suc x) (MTArray x11 l) (MTArray x11 l) destl destl'"
            using True root_bundle unfolding CompMemType.simps by blast
          ultimately show ?thesis using root_bundle mtypes.simps(5) True by simp
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using a2 by blast
          then have "(∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' = Some
tt)"
            using cps2mrec_somesome[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t' ms v'
v''] 2 a30 by simp
          then show ?thesis using root_bundle by auto
        qed
      next
        case (STMap x21 x22)
        then show ?thesis using 2 by simp
      next
        case (STValue x2)
        then have tdef:"t = MTValue x2" using 2(3) cps2mTypeCompatible.simps
          using convertComp by force
        then have "destl' = hash destl (ShowL_nat x)" using a2 2(3) cps2mTypeCompatible.simps
          by (metis TypedMemSubPref.simps(1))
        then have v'':"Some v'' = (let v = accessStorage x2 (hash srcl (ShowL_nat x)) ms in Some
(updateTypedStore (hash
            destl (ShowL_nat x)) (MValue v) (MTValue x2) v'))"
          using a30 cps2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 ms v'
] STValue by simp
        then obtain v where "accessStorage x2 (hash srcl (ShowL_nat x)) ms = (v)" by (auto
split:option.splits memoryvalue.splits)
        then have "Some v'' = Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2)
v')" using v'' by simp
        then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using tdef STValue
          using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def a30 cps2mrec_somesome
          by (metis TypedMemSubPref.simps(1) cps2mTypeCompatible.simps(1))
      qed
    qed
  qed
qed


end
```

12 Environment initialisation and declarations (TypeSafe_Environment)

```
theory TypeSafe_Environment
  imports TypeSafe_Def TypeSafe_Memory_Init TypeSafe_Memory_Copies
begin

context typesafe_base
begin
```

12.1 Initialising an environment results in a type-safe environment

In the call and ecall expressions all calculations are done in a clean environment. To do this `ffold_init` is used to create the environment. The following lemmas demonstrate the initialisation process results in a typesafe environment

12.1.1 Safety of empty environments

```
lemma emptyTopLocs:
  shows "lessThanTopLocs emptyStore" unfolding lessThanTopLocs_def emptyStore_def accessStore_def by
simp

lemma typedEmptyTopLocs:
  shows "lessThanTopLocs emptyTypedStore"
  unfolding lessThanTopLocs_def emptyStore_def emptyTypedStore_def accessStore_def by simp

lemma emptyEnvSafe:
  assumes "ev = emptyEnv addr contrct sendr sval"
    and "safeContract acc stor"
    and "balanceTypes acc"
    and "svalueTypes (Svalue ev)"
    and "lessThanTopLocs sck"
    and "lessThanTopLocs cd"
    and "lessThanTopLocs mem"
    and "addressFormat addr"
    and "addressFormat sendr"
    and "AddressTypes acc"
    and "subPrefixStructuralConsistency mem"
    and "SomeValSomeTyp mem"
  shows "TypeSafe ev acc sck mem stor cd" unfolding TypeSafe_def
proof intros
  have *:"Denvalue (ev) = fmempty" using assms(1) unfolding emptyEnv_def by simp
  then show " unique_locations (Denvalue ev)" unfolding unique_locations_def by simp
  show "AddressTypes acc" using assms by auto
  show " compPointers sck (Denvalue ev)" unfolding compPointers_def using * by (simp split:if_splits)
  show "balanceTypes acc" using assms(3) by simp
  show "svalueTypes (Svalue ev)" using assms(4) by simp
  show "lessThanTopLocs sck" using assms(5) by simp
  show " lessThanTopLocs cd" using assms(6) by simp
  show " lessThanTopLocs mem" using assms(7) by simp
  show "addressFormat (Address ev)" using assms(8) assms(1) by simp
  show "addressFormat (Sender ev)" using assms(9) assms(1) by simp
  show "safeContract acc stor" using assms by simp
  show " typeCompat (Denvalue ev) sck mem (stor (Address ev)) cd " using * unfolding typeCompat_def by
simp
  show "denvalueTypeCorrectness ev sck mem" using * unfolding denvalueTypeCorrectness_def by simp
  show "subPrefixStructuralConsistency mem" using * assms unfolding subPrefixStructuralConsistency_def
by simp
  show "SomeValSomeTyp mem" using assms(12) assms(1) by simp

qed
```

12.1.2 Variable initialisation preserves unique locations

```
lemma envUpdate_other_unchanged:
  assumes "e' = e (|Denvalue := fmupd i ((type.Storage tp),(Storeloc i)) (Denvalue e)|)"
  shows"(∀x. x ≠ i ⟶ fmlookup (Denvalue e') x = fmlookup (Denvalue e) x)"
proof(intros)
  fix x
  assume a1:"x ≠ i"
  show "Denvalue e' $$ x = Denvalue e $$ x" using assms a1 by simp
qed
```

There is an assumption here that the variable being loaded does not already exist in the environment. This is gained from the context of *ffold_init* as we are either loading into an empty environment or if the location already exists then an update function is called so the value is overwritten

```
lemma uniqueLocationsPreserved:
  assumes "unique_locations (Denvalue e)"
  assumes "e' = e (| Denvalue := fmupd i ((type.Storage tp),(Storeloc i)) (Denvalue e) |)"
    and "(∀x y. Denvalue e $$ x = Some y ⟶ snd y ≠ Storeloc i)"
```

```
  shows "unique_locations (Denvalue e')" unfolding unique_locations_def
proof(intros)
  fix x y
  assume a1:"x |∈| fmran (Denvalue e') ∧ y |∈| fmran (Denvalue e') ∧ snd x = snd y"
  have a2:"fmlookup (Denvalue e') i = Some ((type.Storage tp),(Storeloc i))" using assms(2) by simp
  show "x = y"
  proof(cases "Denvalue e' $$ i = Some x")
    case xt:True
    then have a3: "x = ((type.Storage tp),(Storeloc i))" using a2 by simp
    then show ?thesis
    proof(cases "Denvalue e' $$ i = Some y")
      case True
      then show ?thesis using xt by simp
    next
      case False
      then obtain t where "Some y = Denvalue e' $$ t " using assms a1 fmranE by metis
      then have "Denvalue e $$ t = Some y" using assms a1 False by auto
      then have "snd y ≠ snd x" using assms a1 a3 by fastforce
      then show ?thesis using a1 assms by simp
    qed
  next
    case xf:False
    then show ?thesis
    proof(cases "Denvalue e' $$ i = Some y")
      case True
      then obtain t where "Some x = Denvalue e' $$ t " using assms a1 fmranE by metis
      then have "Denvalue e $$ t = Some x" using assms a1 xf by auto
      then have "snd y ≠ snd x" using assms True a1 a2 xf by fastforce
      then show ?thesis using a1 by simp
    next
      case False
      then show ?thesis using False a1 assms envUpdate_other_unchanged fmranE fmranI
unique_locations_def xf by metis
    qed
  qed
qed


lemma compPointersPreserved:
  assumes "compPointers st (Denvalue e)"
  assumes "e' = e (| Denvalue := fmupd i ((type.Storage tp), (Storeloc i)) (Denvalue e) |)"
    and "SCon tp i (stor (Address e'))"
    and "TypeSafe e acc st mem stor cd"
    and "(∀x y. Denvalue e $$ x = Some y ⟶ snd y ≠ Storeloc i)"
    and "∀x y. Denvalue e $$ x = Some y ⟶ snd y = Storeloc x"
    and "∀ x y t1 t2. x ≠ y ∧ Denvalue e $$ x = Some (type.Storage t1, Storeloc x) ∧ Denvalue e $$ y
= Some (type.Storage t2, Storeloc y)
       ⟶ ¬TypedStoSubpref x y t2 ∧ ¬TypedStoSubpref y x t1"
    and "∀x t1. x ≠ i ∧ Denvalue e $$ x = Some (type.Storage t1, Storeloc x) ⟶ ¬TypedStoSubpref x i
tp ∧ ¬TypedStoSubpref i x t1"
  shows "compPointers st (Denvalue e')"
  unfolding compPointers_def
proof (intro allI impI)
  fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
  have a2:"fmlookup (Denvalue e') i = Some ((type.Storage tp),(Storeloc i))" using assms(2) by simp
  have storSame:"stor (Address e') = stor (Address e)" using assms(2) by simp
  assume *: "(type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e') ∧
       (l1 = Stackloc l1' ∧ accessStore l1' st = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2 =
Stackloc l2' ∧ accessStore l2' st = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
  show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
       else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"

  proof(cases "TypedStoSubpref stl2 stl1 tp1")
```

```
    case **:True
    have "CompStoType tp1 tp2 stl1 stl2"
    proof(cases "Denvalue e' $$ i = Some (type.Storage tp1, l1)")
      case t1:True
      then have a3: "(type.Storage tp1, l1) = ((type.Storage tp),(Storeloc i))" using assms(2) by simp
      then show ?thesis
      proof(cases "Denvalue e' $$ i = Some (type.Storage tp2, l2)")
        case True
        then have a4:"(type.Storage tp1, l1) = (type.Storage tp2, l2)" using assms(2) a3 by simp
        then have "tp1 = tp ∧ tp2 = tp" using a3 by simp
        moreover have "l1 = Storeloc i ∧ l2 = Storeloc i" using a3 a4 by simp
        ultimately show ?thesis using assms(3) * CompStoType.simps **
          by (metis denvalue.distinct(1) denvalue.inject(2) stypes.exhaust)
      next
        case False
        then have a5:"(type.Storage tp2, l2) |∈| fmran (Denvalue e)" using assms(1) unfolding
compPointers_def using * assms envUpdate_other_unchanged fmranE fmranI by metis
        then have a10:"l2 ≠ Storeloc i" using * ** t1 assms(5) fmranE by auto
        then have a20:"l1 = Storeloc i" using a3 by simp
        then have a30:"tp = tp1" using a3 by simp
        then have a40:"stl1 = i" using * a20 by simp
        have "∀c. l2 ≠ Stackloc c" using assms(6) a5 by fastforce
        then obtain i' where i'Def: "l2 = Storeloc i' ∧ i' ≠ i ∧ stl2 = i'" using a10 * by auto
        show ?thesis using ** assms(7) assms(8)
          by (metis denvalue.inject(2) i'Def a30 a40 a5 assms(6) fmranE snd_conv)
      qed
    next
      case f2:False
      then have a5:"(type.Storage tp1, l1) |∈| fmran (Denvalue e)" using assms(1) unfolding
compPointers_def using * assms envUpdate_other_unchanged fmranE fmranI by metis
      then show ?thesis
      proof(cases "Denvalue e' $$ i = Some (type.Storage tp2, l2)")
        case True
        then have a3: "(type.Storage tp2, l2) = ((type.Storage tp),(Storeloc i))" using assms(2) by
simp
        then have a20:"l1 ≠ Storeloc i" using a3 a5 * ** assms(5) fmranE by auto
        then have a30:"tp = tp2" using a3 by simp
        then have a40:"stl2 = i" using * a20 using a3 by auto
        have "∀c. l2 ≠ Stackloc c" using assms(6) a5 a3 by auto
        then obtain i' where a50:"l2 = Storeloc i' ∧ i' ≠ i ∧ stl2 = i'" using *
          by (metis "**" denvalue.distinct(1) denvalue.inject(2) a20 a5 assms(6) assms(8) fmranE sndI)
        show ?thesis using ** assms(7) assms(8)
          using a50 a40 by fastforce
      next
        case False
        then have a6:"(type.Storage tp2, l2) |∈| fmran (Denvalue e)" using assms(1) unfolding
compPointers_def using * assms envUpdate_other_unchanged fmranE fmranI by metis
        then obtain x where b10:"Denvalue e $$ x = Some (type.Storage tp2, Storeloc x)" using fmranE
assms(5) by (metis assms(6) snd_conv)
        then obtain y where b20:"Denvalue e $$ y = Some (type.Storage tp1, Storeloc y)" using a5
fmranE assms(5) by (metis assms(6) snd_conv)
        then show ?thesis using ** assms(4) a6 a5 b10 b20 fmranE unfolding TypeSafe_def
compPointers_def using "*" by force
      qed
    qed
    then show ?thesis by (simp add: "**")
  next
    case f2:False
    then show ?thesis
    proof(cases "TypedStoSubpref stl1 stl2 tp2")
      case **:True
      have "CompStoType tp2 tp1 stl2 stl1"
      proof(cases "Denvalue e' $$ i = Some (type.Storage tp1, l1)")
        case t1:True
```

```
        then have a3: "(type.Storage tp1, l1) = ((type.Storage tp),(Storeloc i))" using assms(2) by
simp
        then show ?thesis
        proof(cases "Denvalue e' $$ i = Some (type.Storage tp2, l2)")
          case True
          then have a4:"(type.Storage tp1, l1) = (type.Storage tp2, l2)" using assms(2) a3 by simp
          then have "tp1 = tp ∧ tp2 = tp" using a3 by simp
          moreover have "l1 = Storeloc i ∧ l2 = Storeloc i" using a3 a4 by simp
          ultimately show ?thesis using assms(3) *
            using "**" f2 by auto
        next
          case False
          then have a5:"(type.Storage tp2, l2) |∈| fmran (Denvalue e)"
            using assms(1) unfolding compPointers_def using * assms envUpdate_other_unchanged fmranE
fmranI by metis
          then have a10:"l2 ≠ Storeloc i" using * ** t1 assms(5) fmranE by auto
          then have a20:"l1 = Storeloc i" using a3 by simp
          then have a30:"tp = tp1" using a3 by simp
          then have a40:"stl1 = i" using * a20 by simp
          have "∀c. l2 ≠ Stackloc c" using assms(6) a5 by fastforce
          then obtain i' where i'Def: "l2 = Storeloc i' ∧ i' ≠ i ∧ stl2 = i'" using a10 * by auto
          show ?thesis using ** assms(7) assms(8)
            by (metis denvalue.inject(2) i'Def a40 a5 assms(6) fmranE snd_conv)
        qed
      next
        case f3:False
        then have a5:"(type.Storage tp1, l1) |∈| fmran (Denvalue e)" using assms(1) unfolding
compPointers_def using * assms envUpdate_other_unchanged fmranE fmranI by metis
        then show ?thesis
        proof(cases "Denvalue e' $$ i = Some (type.Storage tp2, l2)")
          case True
          then have a3: "(type.Storage tp2, l2) = ((type.Storage tp),(Storeloc i))" using assms(2) by
simp
          then have a20:"l1 ≠ Storeloc i" using a3 a5 * ** assms(5) fmranE by auto
          then have a30:"tp = tp2" using a3 by simp
          then have a40:"stl2 = i" using * a20 using a3 by auto
          have "∀c. l1 ≠ Stackloc c" using assms(6) a5 by fastforce
          then obtain i' where a50:"l1 = Storeloc i' ∧ i' ≠ i ∧ stl1= i'" using * a40 a20
            by simp
          show ?thesis using ** assms(7) assms(8)
            using a50 a40 by (metis denvalue.inject(2) a30 a5 assms(6) fmranE snd_conv)
        next
          case False
          then have a6:"(type.Storage tp2, l2) |∈| fmran (Denvalue e)" using assms(1) unfolding
compPointers_def using * assms envUpdate_other_unchanged fmranE fmranI by metis
          then obtain x where b10:"Denvalue e $$ x = Some (type.Storage tp2, Storeloc x)" using fmranE
assms(5) by (metis assms(6) snd_conv)
          then obtain y where b20:"Denvalue e $$ y = Some (type.Storage tp1, Storeloc y)" using a5
fmranE assms(5) by (metis assms(6) snd_conv)
          then show ?thesis using ** f2 f3 assms(4) a6 a5 b10 b20 fmranE unfolding TypeSafe_def
compPointers_def using "*" by force
        qed
      qed
      then show ?thesis by (simp add: "**" f2 )
    next
      case False
      then show ?thesis using f2 by simp
    qed
  qed
qed

lemma compPointersNonStackUpd:
  assumes "compPointers sck' (Denvalue ev')"
    and "Denvalue e = Denvalue (ev'(|Denvalue := Denvalue ev'(ip $$:= (t, (Stackloc (ShowL_nat (Toploc
```

```
sck'))))|)"
    and "∀t'. t ≠ type.Storage t'"
    and "k' = push v sck'"
    and "lessThanTopLocs sck'"
    and "TypeSafe ev' (Accounts st) sck' mem' (Storage st) cd'"
    and "Address ev' = Address e"
  shows "compPointers k' (Denvalue e)" unfolding compPointers_def
proof intros
  fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
  assume *:" (type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2 =
Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
  have b5: "fmlookup (Denvalue e) ip = Some (t,(Stackloc (ShowL_nat (Toploc sck'))))" using assms by
simp
  have b6:"∀i''. i'' ≠ ip ⟶ fmlookup (Denvalue e) i'' = fmlookup (Denvalue ev') i''" using assms
fmupd_def by fastforce
  have b7:"∀loc''. loc'' ≠ (ShowL_nat (Toploc sck')) ⟶ accessStore loc'' k' = accessStore loc'' sck'"
    using assms unfolding push_def allocate_def accessStore_def updateStore_def by simp
  have b8:"∀x y. Denvalue ev' $$ x = Some y ⟶ snd y ≠ Stackloc (ShowL_nat (Toploc sck'))"
    using stackLocs_imp_NotDen[of sck' ev' st mem'] assms by auto
  then have b15:"fmlookup (Denvalue e) ip ≠ Some (type.Storage tp1, l1) ∧ fmlookup (Denvalue e) ip ≠
Some (type.Storage tp2, l2)" using b5 assms by simp
  then have b20:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧ (type.Storage tp2, l2) |∈| fmran
(Denvalue ev')"
    using b6 * by (metis fmlookup_ran_iff)
  then have b25:"l1 ≠(Stackloc (ShowL_nat (Toploc sck'))) ∧ l2 ≠ (Stackloc (ShowL_nat (Toploc sck')))"
    using b5 * b8 b20 by fastforce
  then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
  proof(cases "stl1 = stl2")
    case True
    then show ?thesis
    proof(cases "l1")
      case st1:(Stackloc x1)
      then have a1:"accessStore x1 sck' = Some (KStoptr stl1)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
      then show ?thesis
      proof(cases "l2")
        case (Stackloc x2)
        then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
        then show ?thesis using True assms(6) a1 st1 Stackloc unfolding TypeSafe_def compPointers_def
using b20 by blast
      next
        case (Storeloc x2)
        then show ?thesis using True assms(6) a1 st1 unfolding TypeSafe_def compPointers_def using
b20 using "*" by blast
      qed
    next
      case str1:(Storeloc x2)
      then have a1:"stl1 = x2" using * by simp
      then show ?thesis
      proof(cases "l2")
        case (Stackloc x2)
        then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
        then show ?thesis using True assms(6) a1 str1 Stackloc unfolding TypeSafe_def compPointers_def
using b20 by blast
      next
        case (Storeloc x2)
        then show ?thesis using True assms(6) a1 str1 unfolding TypeSafe_def compPointers_def using
b20 using "*" by blast
      qed
```

```
    qed
  next
    case f1:False
    then show ?thesis
    proof(cases "TypedStoSubpref stl2 stl1 tp1")
      case t2:True
      then show ?thesis
      proof(cases "l1")
        case st1:(Stackloc x1)
        then have a1:"accessStore x1 sck' = Some (KStoptr stl1)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
        then show ?thesis
        proof(cases "l2")
          case (Stackloc x2)
          then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
          then show ?thesis using f1 t2 assms(6) a1 st1 Stackloc unfolding TypeSafe_def
compPointers_def using b20 by blast
        next
          case (Storeloc x2)
          then show ?thesis using f1 t2 assms(6) a1 st1 unfolding TypeSafe_def compPointers_def using
b20 using "*" by blast
        qed
      next
        case str1:(Storeloc x2)
        then have a1:"stl1 = x2" using * by simp
        then show ?thesis
        proof(cases "l2")
          case (Stackloc x2)
          then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding TypeSafe_def
compPointers_def using b20 "*" b25 b7 by auto
          then show ?thesis using f1 t2 assms(6) a1 str1 Stackloc unfolding TypeSafe_def
compPointers_def using b20 by blast
        next
          case (Storeloc x2)
          then show ?thesis using f1 t2 assms(6) a1 str1 unfolding TypeSafe_def compPointers_def us-
ing b20 using "*" by blast
        qed
      qed
    next
      case f2:False
      then show ?thesis
      proof(cases " TypedStoSubpref stl1 stl2 tp2 ")
        case t3:True
        then show ?thesis
        proof(cases "l1")
          case st1:(Stackloc x1)
          then have a1:"accessStore x1 sck' = Some (KStoptr stl1)" using assms(6) unfolding
TypeSafe_def compPointers_def using b20 "*" b25 b7 by auto
          then show ?thesis
          proof(cases "l2")
            case (Stackloc x2)
            then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding
TypeSafe_def compPointers_def using b20 "*" b25 b7 by auto
            then show ?thesis using f1 f2 assms(6) a1 st1 Stackloc unfolding TypeSafe_def
compPointers_def using b20 by blast
          next
            case (Storeloc x2)
            then show ?thesis using f1 f2 assms(6) a1 st1 unfolding TypeSafe_def compPointers_def
using b20 using "*" by blast
          qed
        next
          case str1:(Storeloc x2)
          then have a1:"stl1 = x2" using * by simp
```

```
          then show ?thesis
          proof(cases "l2")
            case (Stackloc x2)
            then have "accessStore x2 sck' = Some (KStoptr stl2)" using assms(6) unfolding
TypeSafe_def compPointers_def using b20 "*" b25 b7 by auto
            then show ?thesis using assms(6) a1 str1 Stackloc unfolding TypeSafe_def compPointers_def
using b20 by blast
          next
            case (Storeloc x2)
            then show ?thesis using assms(6) a1 str1 unfolding TypeSafe_def compPointers_def using
b20 using "*" by blast
          qed
        qed
      next
        case t4:False
        then show ?thesis using f1 f2 by simp
      qed
    qed
  qed
qed


lemma ffoldInitAllLocsStorage:
  shows "∀e' x y. ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval) xs = e'
         ⟶ ( (Denvalue e') $$ x = Some y ⟶ (snd y) = Storeloc x )

             "
proof(induct xs)
  case empty
  then show ?case
  proof(intros)
    fix e' x y
    assume **:"ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval) {||} = e'"
    then have ***:"e' = emptyEnv(Address e2) (Contract e) (Sender e3) sval" using
FSet.comp_fun_commute.ffold_empty[OF init_commte] by simp

    assume ****:"Denvalue e' $$ x = Some y "
    have "Denvalue e' = fmempty" using *** by simp
    then have a23:"(Denvalue e') $$ x = None" by simp
    then show "snd y = Storeloc x" using ** **** by simp
  qed
next
  case (insert x xs)
  then have *: "ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval) (finsert x xs) =
    init ct x (ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval) xs)" using
FSet.comp_fun_commute.ffold_finsert[OF init_commte] by simp
  show ?case
  proof (intros)
    fix e' xa y ya assume **: "ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval)
(finsert x xs) = e'"
    obtain e'' where ***: "ffold (init ct) (emptyEnv (Address e2) (Contract e) (Sender e3) sval) xs =
e''" using * by simp
    then have a15: "(Denvalue e'' $$ x = Some y ⟶ snd y = Storeloc x)" using insert * *** by blast
    then have a20:"init ct x e'' = e'" using * ** *** by simp
    then have a23:"Contract e = Contract (emptyEnv (Address e) (Contract e) (Sender e) (Svalue e))"
using a20 by simp
    then have a25: "Contract e = Contract e''" using *** ffold_init_contract by auto
    have "(case ct $$ x of None ⇒ e'' | Some (Var tp) ⇒ updateEnvDup x (type.Storage tp) (Storeloc x)
e'' | Some _ ⇒ e'') = e'"
      using init_def[of ct x e''] a20 by auto


    assume a40:"Denvalue e' $$ xa = Some y"
    show "snd y = Storeloc xa"
```

```
    proof (cases "Denvalue e'' $$ xa = Some y")
      case True
      then show ?thesis using insert *** by blast
    next
      case False
      then have a50:"Denvalue e' ≠ Denvalue e''" using a40 by auto
      then obtain tp where a60:"fmlookup ct x = Some (Var tp)" using a50 a20 a40 init_def by (simp
split:option.splits member.splits )
      then have a70:"Denvalue e' = Denvalue(updateEnvDup x (type.Storage tp) (Storeloc x) e'')" using
a50 a20 a40 by (simp split:option.splits member.splits )
      then have a80:"Denvalue e' = Denvalue(e''(| Denvalue := fmupd x ((type.Storage tp),(Storeloc x))
(Denvalue e'')|))" using
          updateEnvDup.simps updateEnv.simps by (metis a20 a50 a60 init_s12 init_s13)
      then have a90:"xa = x" using a70 a40 a20 a50 a60 init_def updateEnvDup_dup False by metis
      then have "Denvalue e' $$ x = Some ((type.Storage tp),(Storeloc x))" using a80 by simp
      then have "y = ((type.Storage tp), (Storeloc x))" using a90 a40 by simp
      then show ?thesis using a90 by simp
    qed

  qed
qed

lemma DenvalueChange:
  assumes "e' = e(|Denvalue := Denvalue e(i $$:= y)|)"
    and "z ≠ i"
  shows "Denvalue e' $$ z = Denvalue e $$ z" using assms fmranI fmranE by auto
```

12.1.3 Initialising a single variable preserves type safety

```
lemma initEmptySafe:
  assumes "TypeSafe e acc st mem stor cd"
  assumes "init ct i e = e'"
    and "ep $$ Contract (e::environment) = Some(ct, dud)"
    and "Type (acc (Address (e::environment))) = Some (atype.Contract (Contract e))"
    and "∀x y. (Denvalue e) $$ x = Some y ⟶ (snd y) = Storeloc x"
    and "∀l y. ∃t1. (Denvalue e) $$ l = Some y ⟶ ct $$ l = Some (Var t1) ∧ (fst y) = type.Storage
t1"
  shows "TypeSafe e' acc st mem stor cd"
proof (cases "fmlookup ct i")
  case None
  then have "init ct i e = e" using init_def by simp
  then show ?thesis using assms by simp
next
  case (Some a)
  then show ?thesis
  proof(cases a)
    case (Method x1)
    then have "init ct i e = e" using init_def Some by simp
    then show ?thesis using assms by simp
  next
    case (Function x2)
    then show ?thesis using init_def Some assms by simp
  next
    case (Var tp)
    then have a0:"ct $$ i = Some (Var tp)" using Some by auto
    have a1:"∀c adv ct dud i1 i2 t1 t2.
     i1 ≠ i2 ∧ ep $$ c = Some (ct, dud) ∧ ct $$ i1 = Some (Var t1) ∧ ct $$ i2 = Some (Var t2) ⟶
     ¬ TypedStoSubpref i1 i2 t2 ∧ ¬ TypedStoSubpref i2 i1 t1"
      using methodVarsNoPref by blast
    have a5:"Address e = Address e'" using init_def Some assms(2) Var by (auto split:option.split)
    have a51:"Contract e = Contract e'"using init_def Some assms(2) Var by (auto split:option.split)
    then have "init ct i e = updateEnvDup i (type.Storage tp) (Storeloc i) e" using Var init_def Some
by simp
    then have e'Def:"e' = (case fmlookup (Denvalue e) i of
```

```
            Some _ ⇒ e
          | None ⇒ updateEnv i (type.Storage tp) (Storeloc i) e)" using updateEnvDup.simps assms(2)
by simp
    then show ?thesis
    proof (cases "fmlookup (Denvalue e) i")
      case None
      then have a6:"e' = updateEnv i (type.Storage tp) (Storeloc i) e" using updateEnvDup.simps
assms(2) Some Var by simp
      then have a10:"e' = e (| Denvalue := fmupd i ((type.Storage tp),(Storeloc i)) (Denvalue e) |)"
using updateEnv.simps by simp

      have a14:"Type (acc (Address (e'::environment))) = Some (atype.Contract (Contract e'))"
        using assms(4) a5 a51 by simp

      have a15:" (∀ e ct dud i tp.
      Type (acc (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
      ep $$ Contract e = Some (ct, dud) ∧ ct $$ i = Some (Var tp) ⟶
      SCon tp i (stor (Address (e::environment))))"
        using assms(1) unfolding TypeSafe_def safeContract_def by blast

      then have a20:"SCon tp i (stor (Address e'))" using a15 a0 a14 assms(3)
        by (metis a0 a14 a51)

      show ?thesis unfolding TypeSafe_def
      proof(intros)
        show "safeContract acc stor" using assms(1) TypeSafe_def by simp
        show "balanceTypes acc" using assms TypeSafe_def by simp

      next
        have a30:"svalueTypes (Svalue e)" using assms TypeSafe_def by simp
        then have "Svalue e= Svalue e'" using a10 by simp
        show "svalueTypes (Svalue e')" using assms TypeSafe_def svalueTypes_def a30 a10 by simp
      next
        show "typeCompat (Denvalue e') st mem (stor (Address e')) cd" unfolding typeCompat_def
        proof intros
          fix t l
          assume a40:"(t, l) |∈| fmran (Denvalue e')"
          then obtain x where a50:"fmlookup (Denvalue e') x = Some(t,l)" by auto
          show "denvalueCon st mem (stor (Address e')) cd t l"
          proof(cases "x = i")
            case True
            then have a60:"fmlookup (Denvalue e') x = Some ((type.Storage tp),(Storeloc i))" using a10
a50 by simp
            moreover have "l = Storeloc i" using a50 a60 by simp
            moreover have "t = type.Storage tp" using a50 a60 by simp
            ultimately show ?thesis using a20 by simp
          next
            case False
            then have a60:"fmlookup (Denvalue e) x = fmlookup (Denvalue e') x" using
envUpdate_other_unchanged a10 by simp
            then have a70:"(t, l) |∈| fmran (Denvalue e)" using False a40 a50 by (simp add: fmranI)
            then show ?thesis using assms(1) unfolding TypeSafe_def typeCompat_def using a70 a5 a60
by metis
          qed
        qed
      next
        have *:"unique_locations (Denvalue e)" using assms TypeSafe_def by simp
        have **:"(∀ x y. Denvalue e $$ x = Some y ⟶ snd y ≠ Storeloc i)" using None assms by
fastforce
        show "unique_locations (Denvalue e')" using uniqueLocationsPreserved * ** a10 by blast
      next
        have *:"compPointers st (Denvalue e)" using assms TypeSafe_def by simp
        have **:"(∀ x y. Denvalue e $$ x = Some y ⟶ snd y ≠ Storeloc i)" using None assms by
fastforce
```

```
        have a21:"SCon tp i (stor (Address e'))" using a20 by simp
        have "init ct i e = e'" using assms by simp
        moreover have "∀x y t1 t2. x ≠ y ∧ Denvalue e $$ x = Some (type.Storage t1, Storeloc x)
              ∧ Denvalue e $$ y = Some (type.Storage t2, Storeloc y)
              ⟶ ¬ TypedStoSubpref x y t2 ∧ ¬ TypedStoSubpref y x t1"
        proof intros
          fix x y t1 t2
          assume b99:"x ≠ y ∧ Denvalue e $$ x = Some (type.Storage t1, Storeloc x) ∧ Denvalue e $$ y
= Some (type.Storage t2, Storeloc y)"
          then have b98:"ct $$ x = Some (Var t1)" using assms(6) by auto
          moreover have b97:"ct $$ y = Some (Var t2)" using assms(6) b99 by auto
          ultimately show " ¬ TypedStoSubpref x y t2" using a1 assms(3) b99 by metis
          show "¬ TypedStoSubpref y x t1" using a1 assms(3) b99 b98 b97 by metis
        qed

        moreover have "∀x t1. x ≠ i ∧ Denvalue e $$ x = Some (type.Storage t1, Storeloc x) ⟶ ¬
TypedStoSubpref x i tp ∧ ¬ TypedStoSubpref i x t1"
          using assms(5) assms(6) assms(3) assms(1) a1 a0 unfolding TypeSafe_def
          by (metis type.inject(4) fst_conv)
        ultimately show "compPointers st (Denvalue e')" using a6 a10 compPointersPreserved[of st e e'
i tp] assms(1) a21 * ** assms(5) by blast
      next
        show "lessThanTopLocs st" using assms TypeSafe_def by simp
      next
        show "lessThanTopLocs cd" using assms TypeSafe_def by simp
      next
        show "lessThanTopLocs mem" using assms TypeSafe_def by simp
      next
        show "addressFormat (Address e')" using assms unfolding TypeSafe_def by auto
      next
        show "addressFormat (Sender e')" using assms unfolding TypeSafe_def by auto
      next
        show "AddressTypes acc" using assms unfolding TypeSafe_def by blast
      next
        have "(∀x y i''. Denvalue e' $$ x = Some y ⟶ snd y ≠ Stackloc i'')" using assms a10 by auto
        then show "denvalueTypeCorrectness e' st mem"
          using assms unfolding TypeSafe_def denvalueTypeCorrectness_def by fastforce
      next
        show "subPrefixStructuralConsistency mem"
          using assms unfolding TypeSafe_def subPrefixStructuralConsistency_def by fastforce
      next
        show "SomeValSomeTyp mem" using assms unfolding TypeSafe_def by blast
      qed
    next
      case some2:(Some a)
      then have a6:"e' = e" using updateEnvDup.simps assms(2) Some Var by simp
      then show ?thesis using assms by simp
    qed
  qed
qed
```

12.1.4 Initialising multiple variables from an empty environment is type-safe

```
lemma ffoldInitTypeSafe:
  assumes " safeContract acc stor"
    and "balanceTypes acc"
    and "svalueTypes sval"
    and "lessThanTopLocs mem"
    and "ep $$ Contract e = Some(ct, dud)"
    and "addressFormat (e2)"
    and "addressFormat (e3)"
    and "AddressTypes acc"
    and "subPrefixStructuralConsistency mem"
    and "SomeValSomeTyp mem"
```

```
    and "Type (acc e2) = Some (atype.Contract (Contract e))"
  shows "∀e' x y. ffold (init ct) (emptyEnv (e2) (Contract e) (e3) sval) xs = e'
         ⟶ TypeSafe e' acc emptyStore mem stor emptyTypedStore
             ∧( (Denvalue e') $$ x = Some y ⟶ (snd y) = Storeloc x )
             ∧ ( Denvalue e' $$ x = Some y ⟶ (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage
t1))
             "
proof(induct xs)
  case empty
  then show ?case
  proof(intros)
    fix e' x y
    assume **:"ffold (init ct) (emptyEnv (e2) (Contract e) ( e3) sval) {||} = e'"
    then have ***:"e' = emptyEnv(e2) (Contract e) ( e3) sval" using FSet.comp_fun_commute.ffold_empty[OF
init_commte] by simp
    then have "svalueTypes sval" unfolding emptyEnv_def using assms(3) svalueTypes_def by simp
    have " safeContract acc stor" using assms(1) ***
      using safeContract_def by force
    then show "TypeSafe e' acc emptyStore mem stor emptyTypedStore"
      using ** *** assms emptyEnvSafe emptyTopLocs typedEmptyTopLocs emptyEnv_svalue
      by (metis)
    assume a22: "Denvalue e' $$ x = Some y"
    have "Denvalue e' = fmempty" using *** by simp
    then have a23:"(Denvalue e') $$ x = None" by simp
    then show "snd y = Storeloc x" using a22 by simp
    then show " ∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1" using a22 a23 by auto
  qed
next
  case (insert x xs)
  then have *: "ffold (init ct) (emptyEnv (e2) (Contract e) ( e3) sval) (finsert x xs) =
    init ct x (ffold (init ct) (emptyEnv (e2) (Contract e) ( e3) sval) xs)" using
FSet.comp_fun_commute.ffold_finsert[OF init_commte] by simp
  show ?case
  proof (intros)
    fix e' xa y ya assume **: "ffold (init ct) (emptyEnv ( e2) (Contract e) ( e3) sval) (finsert x xs)
= e'"
    obtain e'' where ***: "ffold (init ct) (emptyEnv ( e2) (Contract e) ( e3) sval) xs = e''" using *
by simp
    then have a10:"TypeSafe e'' acc emptyStore mem stor emptyTypedStore" using insert * by simp
    then have a15: "(Denvalue e'' $$ x = Some y ⟶ snd y = Storeloc x)" using insert * *** by blast
    then have a16:" Denvalue e'' $$ xa = Some ya ⟶ (∃t1. ct $$ xa = Some (Var t1) ∧ fst ya =
type.Storage t1)" using insert * *** a10 by blast
    then have a20:"init ct x e'' = e'" using * ** *** by simp
    then have a23:"Contract e = Contract (emptyEnv (Address e) (Contract e) (Sender e) (Svalue e))"
using a20 by simp
    then have a25: "Contract e = Contract e''" using *** ffold_init_contract by auto
    have a26: "Address e'' = e2" using *** ffold_init_ad_same
      by force
    have a27: "Type (acc (Address e'')) = Some (atype.Contract (Contract e''))"
      using assms(11) a25 a26 by simp
    have "(case ct $$ x of None ⇒ e'' | Some (Var tp) ⇒ updateEnvDup x (type.Storage tp) (Storeloc x)
e'' | Some _ ⇒ e'') = e'"
      using init_def[of ct x e''] a20 by auto
    then show a30:"TypeSafe e' acc emptyStore mem stor emptyTypedStore"
      using initEmptySafe[of e'' acc emptyStore mem stor emptyTypedStore ct x] assms(2)
            a10 a15 a20 a25 a16 insert *** assms(5) a27 by metis

    assume a40:"Denvalue e' $$ xa = Some y"
    show "snd y = Storeloc xa"
    proof (cases "Denvalue e'' $$ xa = Some y")
      case True
      then show ?thesis using insert *** by blast
    next
      case False
```

```
      then have a50:"Denvalue e' ≠ Denvalue e''" using a40 by auto
      then obtain tp where a60:"fmlookup ct x = Some (Var tp)" using a50 a20 a40 init_def by (simp
split:option.splits member.splits )
      then have a70:"Denvalue e' = Denvalue(updateEnvDup x (type.Storage tp) (Storeloc x) e'')" using
a50 a20 a40 by (simp split:option.splits member.splits )
      then have a80:"Denvalue e' = Denvalue(e''(| Denvalue := fmupd x ((type.Storage tp),(Storeloc x))
(Denvalue e'')|))" using
          updateEnvDup.simps updateEnv.simps by (metis a20 a50 a60 init_s12 init_s13)
      then have a90:"xa = x" using a70 a40 a20 a50 a60 init_def updateEnvDup_dup False by metis
      then have "Denvalue e' $$ x = Some ((type.Storage tp),(Storeloc x))" using a80 by simp
      then have "y = ((type.Storage tp), (Storeloc x))" using a90 a40 by simp
      then show ?thesis using a90 by simp
    qed
    show "∃t1. ct $$ xa = Some (Var t1) ∧ fst y = type.Storage t1"
    proof (cases "Denvalue e'' $$ xa = Some y")
      case True
      then show ?thesis using insert *** by blast
    next
      case False
      then have a50:"Denvalue e' ≠ Denvalue e''" using a40 by auto
      then obtain tp where a60:"fmlookup ct x = Some (Var tp)" using a50 a20 a40 init_def by (simp
split:option.splits member.splits )
      then have a70:"Denvalue e' = Denvalue(updateEnvDup x (type.Storage tp) (Storeloc x) e'')" using
a50 a20 a40 by (simp split:option.splits member.splits )
      then have a80:"Denvalue e' = Denvalue(e''(| Denvalue := fmupd x ((type.Storage tp),(Storeloc x))
(Denvalue e'')|))" using
          updateEnvDup.simps updateEnv.simps by (metis a20 a50 a60 init_s12 init_s13)
      then have a90:"xa = x" using a70 a40 a20 a50 a60 init_def updateEnvDup_dup False by metis
      then have "Denvalue e' $$ x = Some ((type.Storage tp),(Storeloc x))" using a80 by simp
      then show ?thesis using a90 a60 a80 a60 a40 by auto
    qed
  qed
qed

lemma ffoldInit_var_storage_mapping:
  assumes "ffold (init ct) (emptyEnv adv c adde v') (fmdom ct) = loaded"
  shows "∀id v. ct $$ id = Some (Var v) ⟶ (Denvalue loaded $$ id = Some (type.Storage v, Storeloc
id))" using assms
proof(induct "fmdom ct" rule: fset_induct)
  case empty
  show ?case
  proof(intros)
    fix id v
    assume "ct $$ id = Some (Var v)"
    then have "id |∈| fmdom ct" using fmdomI by metis
    then have "id |∈| {||}" using empty by simp
    then show "Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)" by blast
  qed
next
  case (insert x xs)
  let ?e_start = "emptyEnv adv c (adde) v'"
  have loaded_def: "ffold (init ct) ?e_start (finsert x xs) = loaded" using insert assms by simp
  show "∀id v. ct $$ id = Some (Var v) ⟶ Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)"
  proof intros
    fix id v
    assume "ct $$ id = Some (Var v)"
    then show "Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)" using insert
      using ffold_init_fmdom by force
  qed
qed

lemma ffoldInit_var_storage_mapping_rev:
  assumes "ffold (init ct) (emptyEnv adv c (adde) v') (fmdom ct) = loaded"
  shows "∀id v. (Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)) ⟶ ct $$ id = Some (Var
```

```
v)" using assms
proof(induct "fmdom ct" rule: fset_induct)
  case empty
  show ?case
  proof(intros)
    fix id v
    assume " Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)"
    then show " ct $$ id = Some (Var v)"
      using assms ffold_init_emptyDen_ran by blast
  qed
next
  case (insert x xs)
  let ?e_start = "emptyEnv adv c (adde) v'"

  have loaded_def: "ffold (init ct) ?e_start (finsert x xs) = loaded" using insert assms by simp
  show "∀id v. Denvalue loaded $$ id = Some (type.Storage v, Storeloc id) ⟶ ct $$ id = Some (Var v)"
  proof intros
    fix id v
    assume "Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)"
    then show "ct $$ id = Some (Var v)" using insert
      using ffold_init_fmdom
      by (meson ffold_init_emptyDen_ran)
  qed
qed

lemma ffoldInit_var_storage_mapping_eq:
  assumes "ffold (init ct) (emptyEnv adv c (adde) v') (fmdom ct) = loaded"
  shows "∀id v. (Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)) ⟷ ct $$ id = Some (Var
v)"
  using ffoldInit_var_storage_mapping_rev ffoldInit_var_storage_mapping assms by blast

end

end
```

13 Declarations (TypeSafe_Decl)

```
theory TypeSafe_Decl
  imports TypeSafe_Environment
begin

context typesafe_base
begin
```

13.1 Declarations are type-safe

decl is used by *local.load* to declare the contract variables in the newly created environment. For simple cases (Stack values) decl will copy the value from the source location into the top of the Stack. For more complex data types such as copying a Memory structure the structure is copied recursively using a support function (such as *cpm2mrec*)

```
lemma cdMemLocsToploc:
  assumes "lessThanTopLocs cd'"
    and "l = (ShowL_nat (Toploc cd'))"
    and "∃dud. (dud, c') = allocate cd'"
    and "Toploc c' = Toploc c"
    and "∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cd' = accessStore locs c"
  shows "lessThanTopLocs c" unfolding lessThanTopLocs_def
proof intros
  fix tloc loc
  assume b20:"Toploc c ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
  then have "Toploc c' ≤ tloc" using assms by simp
  then have b30:"Toploc cd' < tloc" using assms unfolding allocate_def by simp
```

```
  then have "ReadL_nat l < tloc" using assms
    by (simp add: Read_Show_nat'_id)
  then have "l ≠ (ShowL_nat tloc)"
    using Read_Show_nat'_id by auto
  then have "¬ LSubPrefL2 l (ShowL_nat tloc)"
    by (metis LSubPrefL2_def assms(2) hash_inequality ShowLNatDot hash_int_prefix
hash_suffixes_associative)
  then have b40:"¬ LSubPrefL2 loc l" using b20 assms(2) ShowLNatDot
    by (metis LSubPrefL2_def assms(2) hash_suffixes_associative hashesInts)
  then have "accessStore loc cd' = None" using assms b20 b30 unfolding lessThanTopLocs_def using
nat_less_le by auto
  then show "accessStore loc c= None" using assms b40 by auto
next
  fix loc y
  assume b20:" accessStore loc c = Some y"
  then show "∃tloc<Toploc c. LSubPrefL2 loc (ShowL_nat tloc)"
  proof(cases "LSubPrefL2 loc (ShowL_nat (Toploc cd'))")
    case True
    have a10:"Toploc cd' < Toploc c'" using assms unfolding allocate_def updateStore_def by simp
    have "LSubPrefL2 loc (ShowL_nat (Toploc cd'))" using True LSubPrefL2_def by simp
    have "∀l. l≠(ShowL_nat (Toploc cd')) ⟶ ¬(LSubPrefL2
        (ShowL_nat (Toploc cd')) l)" using ShowLNatDot unfolding LSubPrefL2_def hash_def
      using subPrefCannotBeInt by auto
    then show ?thesis using True a10 assms
      using LSubPrefL2_def by auto
  next
    case False
    have a20:"((∀tloc loc. Toploc cd' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc cd'
= None)
          ∧ (∀loc y. accessStore loc cd' = Some y ⟶ (∃tloc<Toploc cd'. LSubPrefL2 loc (ShowL_nat
tloc))))"
      using assms lessThanTopLocs_def[of cd'] by blast
    then obtain tloc where tlocdef:"tloc < Toploc cd' ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      by (metis False assms(2) assms(5) b20)
    moreover have "Toploc cd' < Toploc c'" using assms unfolding allocate_def updateStore_def by simp
    ultimately show ?thesis using assms lessThanTopLocs_def[of cd'] b20 tlocdef
      using order.strict_trans by metis
  qed
qed
```

13.1.1 Type safety of *decl*

Given a type-safe source environment, a type-consistent value to copy, and a successful *decl*, the result of *decl* is type-safe.

Postcondition of *decl* without an initial value.

```
abbreviation declNonePost :: "environment ⇒ stack ⇒ calldataT ⇒ memoryT ⇒ 'a state_scheme ⇒
environment ⇒ calldataT ⇒ memoryT ⇒ stack ⇒ bool"
  where "declNonePost ev' sck' cd' mem' st e c m' k' ≡ TypeSafe e (Accounts st) k' m' (Storage st) c

∧ (∀x l. (∄y y'. x = type.Memory y ∨ x = Value y') ∧ (x, l) |∈| fmran(Denvalue e) ⟶ (x,l)|∈|
fmran(Denvalue ev') )
∧ (∀sckl ptr. accessStore sckl k' = Some (ptr) ∧ (∄y y'. ptr = KMemptr y ∨ ptr = KValue y') ⟶
accessStore sckl sck' = Some ptr)
∧ cd' = c
∧ Toploc mem' ≤ Toploc m'
∧ (∀locs v. accessStore locs mem' = Some v ⟶ accessStore locs m' = Some v)
∧ (∀locs t. accessTypeStore locs mem' = Some t ⟶ accessTypeStore locs m' = Some t)
∧ (∀locs. (∃tloc<Toploc mem'. LSubPrefL2 locs (ShowL_nat (tloc)) ∧ accessStore locs mem' = None) ⟶
accessStore locs m' = None)"

context
  fixes c cd cd' cp e ev ev' ip k' m' mem' sck' st tp
  assumes typeSafeDeclNone_hyps: "TypeSafe ev' (Accounts st) sck' mem' (Storage st ) cd'"
```

```
    "decl ip tp None cp cd (Memory st) (Storage st (Address ev)) (cd', mem', sck', ev') = Some (c, m',
k', e)"
begin
```

A value-type variable declared without an initial value: its default value is pushed on the stack.

```
lemma typeSafeDeclNone_1:
  assumes "tp = Value t"
    and "None = None"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (c', m, k, e')"
    and "Some (c, m', k', e) = Some (c', m, astack_dup ip (Value t) (KValue (ival t)) (k, e'))"
  shows "declNonePost ev' sck' cd' mem' st e c m' k'"
proof -
  let ?case = "declNonePost ev' sck' cd' mem' st e c m' k'"
  note 1 = assms
  have tsOld:"TypeSafe e' (Accounts st) k m' (Storage st ) c'"
    using 1 typeSafeDeclNone_hyps by simp
  have a1:"(k', e) = (case Denvalue e' $$ ip of None ⇒ (push (KValue (ival t)) k, updateEnv ip (Value
t) (Stackloc (ShowL_nat (Toploc k))) e')
                  | Some x ⇒ (k, e'))" using 1 unfolding astack_dup.simps by simp
  have sameMem:"m' = mem'" using 1 by blast

  show ?thesis
  proof(cases "Denvalue e' $$ ip")
    case (Some a)
    then have same:"k' = k ∧ e = e'" using a1 by simp
    have ts1:"TypeSafe e (Accounts st) k' m' (Storage st) c"
      using 1(7,8) typeSafeDeclNone_hyps(1)
      using same by fastforce
    moreover have sameMem:"mem' = m" using 1 by blast
    then have "Mapping mem' ⊆_f Mapping m" using fmsubset_def
      by (simp add: fmsubset.rep_eq)
    moreover have "(∀ t id' loc. Denvalue ev' $$ id' = Some (type.Storage t, loc) ⟶ Denvalue e $$ id'
= Some (type.Storage t, loc))"
      using same "1"(7) by force
    moreover have "(∀ loc. accessStore loc mem' ≠ accessStore loc m' ⟶ LSubPrefL2 loc (ShowL_nat
(Toploc mem')))"
      using sameMem 1 by simp
    moreover have "Toploc mem' ≤ Toploc m'" using 1 by blast
    moreover have "(∀ loc v. accessStore loc k' = Some (KStoptr v) ⟶ accessStore loc sck' = Some
(KStoptr v))"
      using same 1 by blast
    moreover have "cd' = c" using 1 by simp
    moreover have "ReachableMem e k' m' = ReachableMem ev' sck' mem'"
      unfolding ReachableMem.simps using 1 sameMem same by blast
    moreover have " AllocatedMem_between mem' m' = {}" using 1 sameMem unfolding
AllocatedMem_between_def by simp
    moreover have "(∀ locs. (∃ tloc<Toploc mem'. LSubPrefL2 locs (ShowL_nat (tloc)) ∧ accessStore
       locs mem' = None) ⟶ accessStore locs m' = None)"
      using sameMem 1 by auto
    ultimately show ?thesis using ts1 same 1 sameMem by blast
  next
    case None
    then have eDef:"updateEnv ip (Value t) (Stackloc (ShowL_nat (Toploc k))) e' = e" using a1 by simp
    have k'Def:"k' = push (KValue (ival t)) k" using a1 None by simp
    have sameMem:"m' = mem'" using 1 by blast
    have sameAddress:"Address e = Address ev'" using eDef 1 by auto
    have sameCd:"c = cd'" using 1 by blast
    have lessTop:"lessThanTopLocs sck'" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by blast
    show ?thesis
    proof(intros)
```

```
      show "TypeSafe e (Accounts st) k' m' (Storage st) c"
        unfolding TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st)" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by
simp
      next
        show "safeContract (Accounts st) (Storage st)" using typeSafeDeclNone_hyps(1) unfolding
TypeSafe_def by simp
      next
        show "unique_locations (Denvalue e)" unfolding unique_locations_def
        proof intros
          fix x y

          assume *:"x |∈| fmran (Denvalue e) ∧ y |∈| fmran (Denvalue e) ∧ snd x = snd y"
          then obtain i1 i2 where i1def:"(Denvalue e) $$ i1 = Some x" and i2def:"Denvalue e $$ i2 =
Some y" by blast
          show "x = y"
          proof(cases "i1 = ip")
            case True
            then have xIs:"x = (Value t, Stackloc (ShowL_nat (Toploc k)))" using i1def eDef unfolding
updateEnv.simps by auto
            then show ?thesis
            proof(cases "i2 = ip")
              case t2:True
              then show ?thesis using True i1def i2def by simp
            next
              case False
              then have "y |∈| fmran (Denvalue e')" using eDef i2def * unfolding updateEnv.simps
                by (metis DenvalueChange fmranI)
              then show ?thesis using stackLocs_imp_NotDen[OF _ typeSafeDeclNone_hyps(1)] 1(7) * xIs
                by (metis fmranE snd_conv stackLocs_imp_NotDen tsOld TypeSafe_def)
            qed
          next
            case False
            then have xOld:"x |∈| fmran (Denvalue e')" using eDef i1def * unfolding updateEnv.simps
              by (metis "1"(7) typeSafeDeclNone_hyps(2) decl_env_not_i fmranI)
            then show ?thesis
            proof(cases "i2 = ip")
              case True
              then have yIs:"y = (Value t, Stackloc (ShowL_nat (Toploc k)))" using i2def eDef unfolding
updateEnv.simps by auto
              then show ?thesis using stackLocs_imp_NotDen[OF _ typeSafeDeclNone_hyps(1)] 1(7) * xOld
                by (metis eq_snd_iff fmranE stackLocs_imp_NotDen tsOld TypeSafe_def)
            next
              case False
              then have "y |∈| fmran (Denvalue e')" using eDef i2def * unfolding updateEnv.simps
                by (metis DenvalueChange fmranI)
              then show ?thesis using xOld * typeSafeDeclNone_hyps(1) 1(7) unfolding TypeSafe_def
unique_locations_def by blast
            qed
          qed
        qed
      next
        show "compPointers k' (Denvalue e)" unfolding compPointers_def
        proof intros
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume in1:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
         (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
         (l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
         (l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then have "(type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧
         (type.Storage tp2, l2) |∈| fmran (Denvalue e')" using eDef unfolding updateEnv.simps

          by (auto split: option.splits memoryvalue.splits mtypes.splits stypes.splits if_splits
```

```
prod.splits stackvalue.splits type.splits
              denvalue.splits simp: None typeSafeDeclNone_hyps(2) decl_env_kval decl_env_not_i
fmlookup_ran_iff)
         moreover have "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ⟶ l1 =
Stackloc l1' ∧ accessStore l1' k = Some (KStoptr stl1)"
           using k'Def in1 unfolding push_def allocate_def updateStore_def accessStore_def by simp
         moreover have "l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ⟶ l2 =
Stackloc l2' ∧ accessStore l2' k = Some (KStoptr stl2)"
           using k'Def in1 unfolding push_def allocate_def updateStore_def accessStore_def by simp
         ultimately show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
        else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
           using typeSafeDeclNone_hyps(1) 1(7) in1 unfolding TypeSafe_def compPointers_def by blast
       qed
     next
       show "balanceTypes (Accounts st)" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by
simp
     next
       show "addressFormat (Address e)" using eDef unfolding updateEnv.simps using 1(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by auto
     next
       show "addressFormat (Sender e)" using eDef unfolding updateEnv.simps using 1(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by auto
     next
       show " svalueTypes (Svalue e)" using eDef unfolding updateEnv.simps using 1(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by auto
     next
       show "lessThanTopLocs k'" unfolding lessThanTopLocs_def
       proof intros
         fix tloc loc
         assume *:"Toploc k' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
         then show "accessStore loc k' = None"
           by (metis k'Def lessThanTopLocs_def stackPushToplocSafe tsOld TypeSafe_def)
       next
         fix loc y
         assume *:"accessStore loc k' = Some y "
         then show "∃tloc<Toploc k'. LSubPrefL2 loc (ShowL_nat tloc)"
           by (metis k'Def lessThanTopLocs_def stackPushToplocSafe tsOld TypeSafe_def)
       qed
     next
       show "lessThanTopLocs c" using 1 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
     next
       show "lessThanTopLocs m'" using 1 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
     next
       show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c "
         unfolding typeCompat_def
       proof intros
         fix t' l'
         assume inDen:"(t', l') |∈| fmran (Denvalue e)"
         then obtain i1 where i1Def:"(Denvalue e) $$ i1 = Some (t', l')" by blast
         show "denvalueCon k' m' (Storage st (Address e)) c t' l'"
         proof(cases l')
           case (Stackloc x1)
           then show ?thesis
           proof(cases "x1 = (ShowL_nat (Toploc k))")
             case True
             then have acc:"accessStore x1 k' = Some (KValue (ival t))"
               using k'Def unfolding push_def allocate_def updateStore_def accessStore_def by simp
             have t'Def:"(Value t) = t'"
             proof(rule ccontr)
               assume "Value t ≠ t'"
               then have "i1 ≠ ip" using i1Def eDef by auto
               then have "(t', l') |∈| fmran (Denvalue e')" using i1Def inDen fmranI
                 by (metis "1"(7) typeSafeDeclNone_hyps(2) decl_env_not_i)
               then show False using Stackloc True stackLocs_imp_NotDen[OF _
```

```
typeSafeDeclNone_hyps(1)] 1(7)
                by (metis fmranE snd_eqD stackLocs_imp_NotDen tsOld TypeSafe_def)
            qed
            have "typeCon t (ival t)"
              by (simp add: ivalTypeCon)
            then show ?thesis using acc Stackloc inDen t'Def by auto
          next
            case False
            then have sameACC:"accessStore x1 k' = accessStore x1 sck'"
              using k'Def 1(7) unfolding push_def allocate_def accessStore_def updateStore_def by
simp
            obtain i where iDef:"(Denvalue e) $$ i = Some (t', l')" using inDen by blast
            then have "i ≠ ip" using eDef False Stackloc by auto
            then have "Denvalue e' $$ i = Some (t', l')" using iDef eDef by auto
            then have inOld:"(t', l') |∈| fmran (Denvalue e')" using fmranI by metis
            have "(case accessStore x1 sck' of None ⇒ False
                  | Some (KValue val) ⇒ (case t' of Value typ ⇒ typeCon typ val | _ ⇒ False)
                  | Some (KCDptr stloc) ⇒ (case t' of Calldata struct ⇒ MCon struct cd' stloc | _ ⇒
False)
                  | Some (KMemptr stloc) ⇒ (case t' of type.Memory struct ⇒ MCon struct mem' stloc |
_ ⇒ False)
                  | Some (KStoptr stloc) ⇒ (case t' of type.Storage struct ⇒ SCon struct stloc
(Storage st (Address ev')) | _ ⇒ False))"
              using inOld sameACC sameMem Stackloc using typeSafeDeclNone_hyps(1) 1(7) unfolding
TypeSafe_def typeCompat_def by fastforce
            then show ?thesis using Stackloc sameACC sameCd sameMem sameAddress
              by (metis denvalue.simps(5))
          qed
        next
          case (Storeloc x2)
          then have inOld:"(t', l') |∈| fmran (Denvalue e')"using eDef inDen unfolding
updateEnv.simps

            by (metis "1"(1,7) None type.distinct(5) typeSafeDeclNone_hyps(2) decl_env_storage
DenvalueChange fmranI i1Def)
          then have "(case t' of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev')) | _ ⇒
False)"
            using typeSafeDeclNone_hyps(1) 1(7) Storeloc inDen sameAddress unfolding TypeSafe_def
typeCompat_def by fastforce
          then obtain struct where "t' = type.Storage struct" by (cases t', simp+)
          then show ?thesis using typeSafeDeclNone_hyps(1) 1(7) Storeloc inDen sameAddress inOld
            unfolding TypeSafe_def typeCompat_def by fastforce
        qed
      qed
    next

      show "denvalueTypeCorrectness e k' m'"
        unfolding denvalueTypeCorrectness_def
      proof intros
        fix tt l ptr_loc
        assume *:"(type.Memory tt, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"
        then obtain i where idef:"Denvalue e $$ i = Some (type.Memory tt, Stackloc l)" by blast
        then have "i ≠ ip" using eDef by auto
        then have inOld:"(type.Memory tt, Stackloc l) |∈| fmran (Denvalue e')"
          using idef eDef * fmranI by fastforce
        then have "l ≠ (ShowL_nat (Toploc k))"
          using k'Def * unfolding push_def updateStore_def allocate_def accessStore_def by auto
        then have "accessStore l k' = accessStore l k"
          using k'Def unfolding push_def updateStore_def allocate_def accessStore_def by simp
        then show "case tt of
    MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)

     | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
```

```
            using inOld typeSafeDeclNone_hyps(1) 1(7,8) *
            unfolding TypeSafe_def denvalueTypeCorrectness_def by auto
        qed
      next
        show "subPrefixStructuralConsistency m'"
          using typeSafeDeclNone_hyps(1) sameMem unfolding TypeSafe_def subPrefixStructuralConsistency_def
by auto
      next
        show "SomeValSomeTyp m'"
          using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def
          using sameMem by blast
      qed
    next
      fix x l
      assume *:"(∄y y'. x = type.Memory y ∨ x = Value y') ∧ (x, l) |∈| fmran (Denvalue e)"
      then obtain ip' where **:"Denvalue e $$ ip' = Some (x, l)" by blast
      then have "ip' ≠ ip" using eDef * unfolding updateEnv.simps by auto
      then have "Denvalue ev' $$ ip' = Some (x, l)"
        using eDef ** decl_env_not_i 1 by force
      then show "(x, l) |∈| fmran (Denvalue ev')" using fmranI by metis
    next
      fix sckl ptr
      assume *:" accessStore sckl k' = Some ptr ∧ (∄y y'. ptr = KMemptr y ∨ ptr = KValue y')"
      have "∃v. accessStore (ShowL_nat (Toploc k)) k' = (Some (KValue v))"
        using k'Def unfolding push_def allocate_def updateStore_def accessStore_def by simp
      then have "sckl ≠ (ShowL_nat (Toploc k))" using * by auto
      then have "accessStore sckl k' =accessStore sckl sck'"
        using k'Def 1 unfolding push_def allocate_def updateStore_def accessStore_def by simp
      then show "accessStore sckl sck' = Some ptr " using * by simp
    next
      show "cd' = c" using 1 by blast
    next
      show "Toploc mem' ≤ Toploc m'" using 1 by blast
    next
      show " ⋀locs v. accessStore locs mem' = Some v ⟹ accessStore locs m' = Some v" using 1 by
blast
    next
      show "⋀locs t. accessTypeStore locs mem' = Some t ⟹ accessTypeStore locs m' = Some t"
        using 1 by blast
    next
      show "⋀locs. ∃tloc<Toploc mem'. LSubPrefL2 locs (ShowL_nat (tloc)) ∧ accessStore locs mem' =
None ⟹ accessStore locs m' = None"
        using 1 by blast
    qed
  qed
qed
```

A memory array declared without an initial value: a fresh array is allocated and initialised with *minit*.

```
lemma typeSafeDeclNone_18:
  assumes "tp = type.Memory (MTArray x t)"
    and "None = None"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (c', m, k, e')"
    and "Some (c, m', k', e) = (if Denvalue e' $$ ip = None ∧ arraysGreaterZero (MTArray x t) then let
m' = minit x t m in Some (c',
        m', astack_dup ip (type.Memory (MTArray x t)) (KMemptr (ShowL_nat (Toploc m))) (k, e')) else
None)"
  shows "declNonePost ev' sck' cd' mem' st e c m' k'"
proof -
  let ?case = "declNonePost ev' sck' cd' mem' st e c m' k'"
  note 18 = assms
```

```
  have tsOld:"TypeSafe e' (Accounts st) k m (Storage st ) c'"
    using 18 typeSafeDeclNone_hyps(1) by simp
  have NoneIP:"Denvalue e' $$ ip = None ∧ arraysGreaterZero (MTArray x t)" using 18 by (simp
split:if_splits)

  then have "(k', e) = (push (KMemptr (ShowL_nat (Toploc m))) k, updateEnv ip (type.Memory (MTArray x
t)) (Stackloc
      (ShowL_nat (Toploc k))) e')" using 18 by simp
  then have k'Def:"k' = push (KMemptr (ShowL_nat (Toploc m))) k" and eDef:"e = updateEnv ip
(type.Memory (MTArray x t)) (Stackloc (ShowL_nat (Toploc k))) e'"
    by blast+
  have m'Def:"m' = minit x t m" using 18 by (auto split:if_split_asm)
  then obtain mm where mmDef:"mm = iter (λi. minitRec (hash ((ShowL_nat (Toploc m))) (ShowL_nat i)) t) m
x" unfolding minit_def by blast
  then have topMM:"Toploc mm = Toploc m" using minitToploc mmDef by metis
  have m'Def2:"snd (allocate mm) = m'" using m'Def unfolding minit_def using mmDef by metis

  have sameAddress:"Address e = Address ev'" using eDef 18 by auto
  have sameCd:"c = cd'" using 18 by (simp split:if_splits)
  have lessTop:"lessThanTopLocs sck'" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by blast
  have mLim:"((∀tloc loc. Toploc m ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc m =
None) ∧
     (∀loc y. accessStore loc m = Some y ⟶ (∃tloc<Toploc m. LSubPrefL2 loc (ShowL_nat tloc))))"
    using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def using 18(7) by blast
  then have noneTop:"∀l. TypedMemSubPref l (ShowL_nat (Toploc m)) (MTArray x t) ∨ l =
     (ShowL_nat (Toploc m)) ⟶ accessStore l m = None"
    by (meson LSubPrefL2_def le_refl typedPrefix_imp_SubPref)
  then have SameAccM_mm:"∀l accx. accessStore l m = Some accx ⟶ accessStore l mm = Some accx"
    using minitSomeOldNew[OF mmDef[symmetric] ] by blast
  have mM'Acc:"∀l accx. accessStore l m = Some accx ⟶ accessStore l m' = Some accx"
  proof intros
    fix l accx
    assume *:"accessStore l m = Some accx "
    have "accessStore (ShowL_nat (Toploc m)) m = None" using mLim
      using LSubPrefL2_def by blast
    then show "accessStore l m' = Some accx" using * m'Def SameAccM_mm mmDef unfolding minit_def
      by (metis allocateSameAccess)
  qed

  have selfPoint:"∀l l'. TypedMemSubPref l (ShowL_nat (Toploc m)) (MTArray x t)
                  ∧ accessStore l m' = Some (MPointer l') ⟶ l' = l"
  proof intros
    have in5:"∀l l'. TypedMemSubPref l (ShowL_nat (Toploc m)) (MTArray x t) ∧ accessStore l mm = Some
(MPointer l') ⟶ l' = l"
      using minitSelfPointers[OF mmDef[symmetric]] by blast
    fix l l'
    assume *:"TypedMemSubPref l (ShowL_nat (Toploc m)) (MTArray x t) ∧ accessStore l m' = Some
(MPointer l')"
    have sameAcc:"∀l. accessStore l m' = accessStore l mm" using m'Def allocateSameAccess mmDef un-
folding minit_def by metis
    then show "l' = l " using in5 * by metis
  qed

  then have prefPtrs_imps_Pref:"∀l. TypedMemSubPrefPtrs m' x t (ShowL_nat (Toploc m)) l ⟶
     TypedMemSubPref l (ShowL_nat (Toploc m)) (MTArray x t)"
    using selfPoint_imps_TypedMemSubPref by blast

  have nonLocChanged:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc m)) (MTArray x t) ∨ locs =
     (ShowL_nat (Toploc m)) ⟶ accessStore locs m = accessStore locs m'"
    using minitSingleChange_typedPref[OF mmDef[symmetric]] m'Def allocateSameAccess mmDef unfolding
minit_def by metis
  have nonLocChanged2:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc m)) ∨ locs =
     (ShowL_nat (Toploc m)) ⟶ accessStore locs m = accessStore locs m'"
    using minitSingleChange[OF mmDef[symmetric]] allocateSameAccess m'Def2 by auto
```

```
  have nonLocChanged2_Typed:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc m)) ∨ locs =
      (ShowL_nat (Toploc m)) ⟶ accessTypeStore locs m = accessTypeStore locs m'"
    using minitSingleChange_typed[OF mmDef[symmetric]] allocateTypeSameAccess m'Def2 by auto
  have nonLocChanged_Typed:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc m)) (MTArray x t) ∨ locs =
      (ShowL_nat (Toploc m)) ⟶ accessTypeStore locs m = accessTypeStore locs m'"
    using minitSingleChange_typedPref_typed[OF mmDef[symmetric]] allocateTypeSameAccess m'Def2 by auto
  have mcTop:"MCon (MTArray x t) mm (ShowL_nat (Toploc m))"
    using MConMinit[of "(ShowL_nat (Toploc m))" t m x mm] mmDef noneTop NoneIP by metis
  then have mcTop:"MCon (MTArray x t) m' (ShowL_nat (Toploc m))" using allocateMCon m'Def2 by blast
  have mcTypes:"∀ destl'.
       TypedMemSubPref destl' (ShowL_nat (Toploc m)) (MTArray x t) ⟶
       (∃ st. CompMemType mm x t st (ShowL_nat (Toploc m)) destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) mm = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' mm = Some (MTValue pval)))"
    using minit_TypeCompChange[OF mmDef[symmetric]] m'Def allocateTypeSameAccess allocateSameAccess
      mmDef CompMemType_SameAccessAllocate unfolding minit_def by blast

  have mcTypes:"∀ destl'.
       TypedMemSubPref destl' (ShowL_nat (Toploc m)) (MTArray x t) ⟶
       (∃ st. CompMemType m' x t st (ShowL_nat (Toploc m)) destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval)))"
  proof intros
    fix destl'
    assume in1:"TypedMemSubPref destl' (ShowL_nat (Toploc m)) (MTArray x t)"
    then obtain st where stDef:"CompMemType mm x t st (ShowL_nat (Toploc m)) destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) mm = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' mm = Some (MTValue pval))"
      using mcTypes by blast
    then have "CompMemType m' x t st (ShowL_nat (Toploc m)) destl'"
      using CompMemType_SameAccessAllocate allocateSameAccess m'Def2 mcTypes by metis
    moreover have "(case st of MTArray parent_len parent_arr ⇒
              ∀ i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval))"
      using m'Def2 allocateTypeSameAccess[of mm] mcTypes in1 stDef
      by presburger
    ultimately show "∃ st. CompMemType m' x t st (ShowL_nat (Toploc m)) destl' ∧
            (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m' = Some parent_arr
             | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval))" by auto
  qed

  have immediateIndex:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc m)) (ShowL_nat i)) m' = Some t"
    using minit_TypeCompChangeIndexs[OF mmDef[symmetric]]
    by (metis allocateTypeSameAccess m'Def2)


  have a120:"lessThanTopLocs sck'" using typeSafeDeclNone_hyps(1) TypeSafe_def 18(7) by simp
  then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
    using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
  then have a140:"∀ x y. ¬((Denvalue ev') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
    using TypeSafe_def typeSafeDeclNone_hyps(1) typeSafeAllStacklocsExist fmranI 18(7) by fastforce
  then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck')))
⟶ ip''' = ip"
    using eDef lessThanTopLocs_def fmranI 18(7) by auto

  show ?thesis
  proof(intros)
    show "TypeSafe e (Accounts st) k' m' (Storage st) c"
      unfolding TypeSafe_def
```

```
    proof intros
      show "AddressTypes (Accounts st)" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show "safeContract (Accounts st) (Storage st)" using typeSafeDeclNone_hyps(1) unfolding
TypeSafe_def by simp
    next
      show "unique_locations (Denvalue e)" unfolding unique_locations_def
      proof intros
        fix x' y
        assume *:"x' |∈| fmran (Denvalue e) ∧ y |∈| fmran (Denvalue e) ∧ snd x' = snd y"
        then obtain i1 i2 where i1def:"(Denvalue e) $$ i1 = Some x'" and i2def:"Denvalue e $$ i2 =
Some y" by blast
        show "x' = y"
        proof(cases "i1 = ip")
          case True
          then have xIs:"x' = (type.Memory (MTArray x t), Stackloc (ShowL_nat (Toploc k)))" using i1def
eDef unfolding updateEnv.simps by simp
          then show ?thesis
          proof(cases "i2 = ip")
            case t2:True
            then show ?thesis using True i1def i2def by simp
          next
            case False
            then have "y |∈| fmran (Denvalue e')" using eDef i2def * unfolding updateEnv.simps
              by (simp add: fmranI)
            then show ?thesis using stackLocs_imp_NotDen[OF lessTop typeSafeDeclNone_hyps(1)] 18(7) *
xIs by fastforce
          qed
        next
          case False
          then have xOld:"x' |∈| fmran (Denvalue e')" using eDef i1def * unfolding updateEnv.simps
            by (simp add: fmranI)
          then show ?thesis
          proof(cases "i2 = ip")
            case True
            then have yIs:"y = (type.Memory (MTArray x t), Stackloc (ShowL_nat (Toploc k)))" using
i2def eDef unfolding updateEnv.simps by simp
            then show ?thesis using stackLocs_imp_NotDen[OF lessTop typeSafeDeclNone_hyps(1)] 18(7) *
xOld by fastforce
          next
            case False
            then have "y |∈| fmran (Denvalue e')" using eDef i2def * unfolding updateEnv.simps
              by (simp add: fmranI)
            then show ?thesis using xOld * typeSafeDeclNone_hyps(1) 18(7) unfolding TypeSafe_def
unique_locations_def by blast
          qed
        qed
      qed
    next
      show "compPointers k' (Denvalue e)" unfolding compPointers_def
      proof intros
        fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
        assume in1:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
        then have "(type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e')" using eDef unfolding updateEnv.simps
          by (auto split: option.splits memoryvalue.splits mtypes.splits stypes.splits if_splits
prod.splits stackvalue.splits type.splits
              denvalue.splits simp: NoneIP typeSafeDeclNone_hyps(2) decl_env_not_i decl_env_storage
fmranE fmranI)

        moreover have "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ⟶ l1 = Stackloc
```

```
l1' ∧ accessStore l1' k = Some (KStoptr stl1)"
          using k'Def in1 unfolding push_def allocate_def updateStore_def accessStore_def by simp
        moreover have "l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ⟶ l2 = Stackloc
l2' ∧ accessStore l2' k = Some (KStoptr stl2)"
          using k'Def in1 unfolding push_def allocate_def updateStore_def accessStore_def by simp
        ultimately show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
       else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
          using typeSafeDeclNone_hyps(1) 18(7) in1 unfolding TypeSafe_def compPointers_def by blast
      qed
    next
      show "balanceTypes (Accounts st)" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show "addressFormat (Address e)" using eDef unfolding updateEnv.simps using 18(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show "addressFormat (Sender e)" using eDef unfolding updateEnv.simps using 18(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show " svalueTypes (Svalue e)" using eDef unfolding updateEnv.simps using 18(7)
typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show "lessThanTopLocs k'" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume *:"Toploc k' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then show "accessStore loc k' = None"
          by (metis k'Def lessThanTopLocs_def stackPushToplocSafe tsOld TypeSafe_def)
      next
        fix loc y
        assume *:"accessStore loc k' = Some y "
        then show "∃tloc<Toploc k'. LSubPrefL2 loc (ShowL_nat tloc)"
          by (metis k'Def lessThanTopLocs_def stackPushToplocSafe tsOld TypeSafe_def)
      qed
    next
      show "lessThanTopLocs c" using NoneIP 18 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by simp
    next
      show "lessThanTopLocs m'" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume *:"Toploc m' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have "Toploc m' > Toploc m" using topMM m'Def2 unfolding allocate_def by auto
        then have "¬LSubPrefL2 loc (ShowL_nat (Toploc m))" using *
          by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general hash_inequality
hash_suffixes_associative hashesIntSame nat_less_le
              order_antisym_conv)
        then have "¬ TypedMemSubPref loc (ShowL_nat (Toploc m)) (MTArray x t) ∨ loc = (ShowL_nat (Toploc
m))"
          using neg_MemLSubPrefL2_imps_TypedMemSubPref by blast
        then show "accessStore loc m' = None"
          using 18 typeSafeDeclNone_hyps(1) nonLocChanged unfolding TypeSafe_def lessThanTopLocs_def
          by (metis "*" <Toploc m < Toploc m'> dual_order.trans mLim nat_less_le)
      next
        fix loc y
        assume *:"accessStore loc m' = Some y"
        then have "Toploc m' > Toploc m" using topMM m'Def2 unfolding allocate_def by auto
        then show "∃tloc<Toploc m'. LSubPrefL2 loc (ShowL_nat tloc)" using *
          by (metis mLim neg_MemLSubPrefL2_imps_TypedMemSubPref nonLocChanged order_less_trans)
      qed
    next
      show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c "
        unfolding typeCompat_def
      proof intros
        fix t' l'
        assume inDen:"(t', l') |∈| fmran (Denvalue e)"
```

```
        then obtain i1 where i1Def:"(Denvalue e) $$ i1 = Some (t', l')" by blast
        show "denvalueCon k' m' (Storage st (Address e)) c t' l'"
        proof(cases l')
          case (Stackloc x1)
          then show ?thesis
          proof(cases "x1 = (ShowL_nat (Toploc k))")
            case True
            then have acc:"accessStore x1 k' = Some (KMemptr (ShowL_nat (Toploc m)))"
              using k'Def unfolding push_def allocate_def updateStore_def accessStore_def by auto
            have t'Def:"t' = (type.Memory (MTArray x t))" using eDef inDen True Stackloc i1Def a150
              using "18"(7) by fastforce
            have "MCon (MTArray x t) m' (ShowL_nat (Toploc m))" using mcTop by simp
            then show ?thesis using Stackloc acc t'Def by simp
          next
            case False
            then have sameACC:"accessStore x1 k' = accessStore x1 sck'"
              using k'Def 18(7) unfolding push_def allocate_def accessStore_def updateStore_def by simp
            obtain i where iDef:"(Denvalue e) $$ i = Some (t', l')" using inDen by blast
            then have "i ≠ ip" using eDef False Stackloc by auto
            then have "Denvalue e' $$ i = Some (t', l')" using iDef eDef by simp
            then have inOld:"(t', l') |∈| fmran (Denvalue e')" using fmranI by metis
            have ol:"(case accessStore x1 sck' of None ⇒ False
                  | Some (KValue val) ⇒ (case t' of Value typ ⇒ typeCon typ val | _ ⇒ False)
                  | Some (KCDptr stloc) ⇒ (case t' of Calldata struct ⇒ MCon struct cd' stloc | _ ⇒
False)
                  | Some (KMemptr stloc) ⇒ (case t' of type.Memory struct ⇒ MCon struct mem' stloc | _
⇒ False)
                  | Some (KStoptr stloc) ⇒ (case t' of type.Storage struct ⇒ SCon struct stloc
(Storage st (Address ev')) | _ ⇒ False))"
              using inOld sameACC sameAddress Stackloc using typeSafeDeclNone_hyps(1) 18(7) unfolding
TypeSafe_def typeCompat_def by fastforce
            then show ?thesis
            proof(cases "accessStore x1 k'")
              case None
              then show ?thesis using inOld sameACC Stackloc ol by simp
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1)
                then show ?thesis using Some inOld sameACC Stackloc ol by simp
              next
                case (KCDptr x2)
                then show ?thesis using Some inOld sameACC Stackloc ol 18(7) sameCd by auto
              next
                case (KMemptr x3)
                then obtain struct where structdef:"t' = type.Memory struct"
                  using ol Some sameACC by (cases t', simp+)
                then have lims:"¬ LSubPrefL2 x3 (ShowL_nat (Toploc m)) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
m)) x3"
                  using typeSafeAllPtrsNotTop[OF typeSafeDeclNone_hyps(1), of struct x1 x3] Stackloc
inOld 18(7) sameACC Some KMemptr by simp
                have lim2:"∀ len arr loc. struct = MTArray len arr ∧ TypedMemSubPrefPtrs mem' len arr
x3 loc
                        ⟶ ¬ LSubPrefL2 loc (ShowL_nat (Toploc m)) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
m)) loc"
                  using typeSafeAllMemPtrsCantTop[OF typeSafeDeclNone_hyps(1), of struct x1 x3]
structdef
                    Stackloc inOld 18(7) sameACC Some KMemptr by auto
                have mcO:"MCon struct mem' x3" using ol Some KMemptr structdef sameACC by simp
                then have "MCon struct m' x3" using MCon_memory_transfer[OF mcO lims lim2, of m']
18(7) nonLocChanged2 by simp
                then show ?thesis using sameAddress Some inOld sameACC Stackloc ol 18(7) KMemptr
structdef by simp
```

```
            next
              case (KStoptr x4)
              then show ?thesis using sameAddress Some inOld sameACC Stackloc ol 18(7) by (cases
t',simp+)
            qed
          qed
        qed
      next
        case (Storeloc x2)
        then have inOld:"(t', l') |∈| fmran (Denvalue e')"using eDef inDen unfolding updateEnv.simps

          using "18"(7) typeSafeDeclNone_hyps(2) decl_env_not_i fmlookup_ran_iff by fastforce
        then have "(case t' of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev')) | _ ⇒
False)"
          using typeSafeDeclNone_hyps(1) 18(7) Storeloc inDen sameAddress unfolding TypeSafe_def
typeCompat_def by fastforce
        then obtain struct where "t' = type.Storage struct" by (cases t', simp+)
        then show ?thesis using typeSafeDeclNone_hyps(1) 18(7) Storeloc inDen sameAddress inOld
          unfolding TypeSafe_def typeCompat_def by fastforce
      qed
    qed
  next
    show "denvalueTypeCorrectness e k' m'"
      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t2 l2 ptr_loc
      assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue e) ∧ accessStore l2 k' = Some
(KMemptr ptr_loc)"
      then obtain i where idef:"Denvalue e $$ i = Some (type.Memory t2, Stackloc l2)" by blast
      show "case t2 of
     MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
      proof(cases "l2 = (ShowL_nat (Toploc k))")
        case True
        then have acc:"accessStore l2 k' = Some (KMemptr (ShowL_nat (Toploc m)))"
          using k'Def unfolding push_def allocate_def updateStore_def accessStore_def by auto
        then have "ptr_loc = (ShowL_nat (Toploc m))" using * by simp
        have t'Def:"t2 = (MTArray x t)" using * eDef True a150
          using "18"(7) by fastforce
        have "∀ i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t"
          by (metis ‹ptr_loc = (ShowL_nat (Toploc m))› allocateTypeSameAccess m'Def2
minit_TypeCompChangeIndexs mmDef)
        then show ?thesis using t'Def ‹ptr_loc = (ShowL_nat (Toploc m))›
          using nonLocChanged noneTop by force
      next
        case False
        then have sameACC:"accessStore l2 k' = accessStore l2 sck'"
          using k'Def 18(7) unfolding push_def allocate_def accessStore_def updateStore_def by simp

        obtain i where iDef:"(Denvalue e) $$ i = Some (type.Memory t2, Stackloc l2)" using * by
blast
        then have "i ≠ ip" using eDef False by auto
        then have inold:"Denvalue e' $$ i = Some (type.Memory t2, Stackloc l2)" using iDef eDef by
simp
        then have ptrDef:"accessStore l2 sck' = Some (KMemptr ptr_loc)" using 18(7) sameACC
          using "*" by auto
        then have old:"(case t2 of
                      MTArray len arr ⇒
          (∀ i<len.
             accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' = Some arr)
         | MTValue val ⇒ accessTypeStore ptr_loc mem' = Some (MTValue val))"
          using typeSafeDeclNone_hyps(1) 18(7) inold
          unfolding TypeSafe_def denvalueTypeCorrectness_def
          by (meson ‹i ≠ ip› typeSafeDeclNone_hyps(2) decl_env_not_i fmranI iDef)
```

```
          have notSubTop:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc m))"
            using ptrDef "18"(8) False ‹Denvalue e' $$ i = Some (type.Memory t2, Stackloc l2)›
decl.simps(18) decl_stack_change fmranI sameACC tsOld
              typeSafeAllPtrsNotTop
            by (metis dual_order.refl)
          then show ?thesis
          proof(cases t2)
            case (MTArray x11 x12)
            have cc1:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc m)) ⟶ accessTypeStore loc m' =
accessTypeStore loc mem'"
              using nonLocChanged2 old nonLocChanged2_Typed 18(7) by simp
            have old': "∀i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' = Some x12"
              using old MTArray by simp

            then show ?thesis using MTArray notSubTop cc1 old' MemLSubPrefL2_specific_imps_general
            proof -
              have "∀i. ¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) (ShowL_nat (Toploc m))"
                using notSubTop MemLSubPrefL2_specific_imps_general by blast
              then show ?thesis using MTArray cc1 old' by simp
            qed
          next
            case (MTValue x2)
            then show ?thesis using notSubTop nonLocChanged2_Typed old 18(7) by simp
          qed
        qed
      qed
    next
      show "subPrefixStructuralConsistency m'" unfolding subPrefixStructuralConsistency_def
      proof(intros)
        fix locs tp
        assume in1:"accessTypeStore locs m' = Some tp"
        have tloc:"Toploc m' > Toploc m" using topMM m'Def2 unfolding allocate_def by auto

        show "case accessStore locs m' of None ⇒ False | Some (MValue v) ⇒ ∃val. MCon tp m' locs ∧
tp = MTValue val ∧ accessTypeStore locs m' = Some tp
       | Some (MPointer p) ⇒ ∃len arr.
             MCon tp m' p ∧
             tp = MTArray len arr ∧
             (∀i<len. accessTypeStore (hash p (ShowL_nat i)) m' = Some arr) "
        proof(cases "accessTypeStore locs mem' = Some tp")
          case True
          then have accL:"∃v. accessStore locs mem' = Some v"
            using typeSafeDeclNone_hyps(1) 18(7) unfolding TypeSafe_def SomeValSomeTyp_def by simp
          have notSub:"¬ LSubPrefL2 locs (ShowL_nat (Toploc m))"
          proof
            assume c:"LSubPrefL2 locs (ShowL_nat (Toploc m))"
            then have "accessStore locs mem' = None"
              using mLim 18(7) tloc
              using nless_le by auto
            then show False using accL by auto
          qed
          then have sameACC:" accessStore locs m' = accessStore locs mem'" using nonLocChanged2 18(7)
by simp
          then consider (ptr) p where "accessStore locs m' = Some (MPointer p)"
            | (val) val2 where "accessStore locs m' = Some (MValue val2)"
            using typeSafeDeclNone_hyps(1) 18(7) True unfolding TypeSafe_def
subPrefixStructuralConsistency_def
            by (metis memoryvalue.exhaust accL)
          then show ?thesis
          proof(cases)
            case ptr
            then obtain len arr where
              old:"MCon tp mem' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat
i)) mem' = Some arr)
```

```
                "
              using typeSafeDeclNone_hyps(1) 18(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
            then have lim2:"∀ len arr loc. tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem' len arr p
loc
                        ⟶ ¬ LSubPrefL2 loc (ShowL_nat (Toploc m)) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc m))
loc"
              using AccessedMemPtrsCantTop[of "mem'" tp p] typeSafeDeclNone_hyps(1) 18(7) unfolding
TypeSafe_def by blast
            have lim:" ¬ LSubPrefL2 p (ShowL_nat (Toploc m)) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc m)) p"
              using AllPtrsNotTop_MCon notSub ptr using typeSafeDeclNone_hyps(1) 18(7) old unfolding
TypeSafe_def by blast
            have accTLoc:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc m)) ∨ loc =
                (ShowL_nat (Toploc m)) ⟶ accessStore loc m' = accessStore loc mem'"
              using 18(7) nonLocChanged2 by auto
            have "MCon tp m' p" using MCon_memory_transfer[OF _ lim lim2 accTLoc] old by blast
            moreover have "∀ i<len. ¬ LSubPrefL2 (hash p (ShowL_nat i)) (ShowL_nat (Toploc m))" using
lim
              using MemLSubPrefL2_specific_imps_general by blast
            ultimately have " MCon tp m' p ∧ tp = MTArray len arr ∧ (∀ i<len. accessTypeStore (hash p
(ShowL_nat i)) m' = Some arr)"
              using nonLocChanged2_Typed old 18(7) by simp

            then show ?thesis using ptr by fastforce
          next
            case val
            then have old:"∃ val. MCon tp mem' locs ∧ tp = MTValue val ∧ accessTypeStore locs mem' =
Some tp"
              using typeSafeDeclNone_hyps(1) 18(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
            then have "∃ val. MCon tp m' locs ∧ tp = MTValue val ∧ accessTypeStore locs m' = Some tp"
              using nonLocChanged2_Typed MCon.simps val notSub 18(7) sameACC by fastforce
            then show ?thesis using val by auto
          qed
        next
          case False
          then have sub:"TypedMemSubPref locs (ShowL_nat (Toploc m)) (MTArray x t) ∧ locs ≠ (ShowL_nat
(Toploc m))"
            using nonLocChanged_Typed 18(7) in1 by fastforce
          then have "TypedMemSubPref locs (ShowL_nat (Toploc m)) (MTArray x t)"
            using prefPtrs_imps_Pref by simp
          then obtain st where stD:"(CompMemType m' x t st (ShowL_nat (Toploc m)) locs ∧
           (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash locs
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore locs m' = Some (MTValue pval)))" using mcTypes by blast
          then have cmp:"CompMemType m' x t st (ShowL_nat (Toploc m)) locs " by blast

          have mcLocs:"MCon st m' locs" using cmp mcTop CompTypeRemainsMCon by auto

          show ?thesis
          proof(cases st)
            case (MTArray x11 x12)
            have subs:"t = MTArray x11 x12 ∧ (∃ i<x. accessStore (hash (ShowL_nat (Toploc m)) (ShowL_nat
i)) m' = Some (MPointer locs)) ∨
                      (∃ midP subL subA i. CompMemType m' x t (MTArray subL subA) (ShowL_nat (Toploc m))
midP
                      ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧
subA = MTArray x11 x12)"
              using CompMemType_imps_Mid[of m' x t x11 x12 "(ShowL_nat (Toploc m))" locs] cmp MTArray by
simp

            then have acc:"accessStore locs m' = Some (MPointer locs)" using selfPoint mcLocs sub
MTArray
              by (metis (no_types, lifting) MCon.simps(2) MCon_imps_TypedMemSubPref_Some mcTop
```

```
option.distinct(1))

            moreover have "∃len arr. MCon tp m' locs ∧ tp = MTArray len arr ∧ (∀i<len.
accessTypeStore (hash locs (ShowL_nat i)) m' = Some arr)
                           ∧ (∀v. accessStore locs m' = Some (MPointer v) ⟶ accessTypeStore locs m'
= Some (MTArray len arr))"
              using in1
            proof(cases "(∃i<x. accessStore (hash (ShowL_nat (Toploc m)) (ShowL_nat i)) m' = Some
(MPointer locs) ∧ t = MTArray x11 x12)")
              case True
              then obtain i where idef:"i<x ∧ accessStore (hash (ShowL_nat (Toploc m)) (ShowL_nat i))
m' = Some (MPointer locs)
                                    ∧ t = MTArray x11 x12"
                by auto
              then have "TypedMemSubPref (hash (ShowL_nat (Toploc m)) (ShowL_nat i)) (ShowL_nat (Toploc
m)) (MTArray x t) " by auto
              then have locsD:"locs = (hash (ShowL_nat (Toploc m)) (ShowL_nat i))" using selfPoint idef
by blast
              have "∀i<x. accessTypeStore (hash (ShowL_nat (Toploc m)) (ShowL_nat i)) m' = Some (MTArray
x11 x12)" using idef
                by (metis allocateTypeSameAccess m'Def2 minit_TypeCompChangeIndexs mmDef)
              then have "tp = MTArray x11 x12" using in1 locsD idef by simp
              moreover have "(∀i<x11. accessTypeStore (hash locs (ShowL_nat i)) m' = Some x12)" using
stD MTArray by simp
              ultimately show ?thesis using mcLocs MTArray
                using in1 by blast
            next
              case False
              then obtain midP i subL subA where mid:"(CompMemType m' x t (MTArray subL subA)
(ShowL_nat (Toploc m)) midP
                   ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
                using subs by blast
              then have locD:"locs = (hash midP (ShowL_nat i))" using selfPoint
                using CompMemType_imps_in_GetAllMemoryLocations_ptr mcTop memSet_selfPoint by blast
              then obtain st2 where st2D:"(CompMemType m' x t st2 (ShowL_nat (Toploc m)) midP ∧
             (case st2 of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash midP
(ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore midP m' = Some (MTValue pval)))" using mid mcTypes
                using CompMemTypeSameLocsSameType CompMemType_imps_TypedMemSubPrefPtrs
prefPtrs_imps_Pref by blast
              then have "st2 = (MTArray subL subA)" using mid CompMemTypeSameLocsSameType mcTop by
blast
              then have " ∀i<subL. accessTypeStore (hash midP (ShowL_nat i)) m' = Some subA" using st2D
by simp
              then have "MTArray x11 x12 = tp" using in1 locD mid by simp
              then show ?thesis using mcLocs MTArray stD in1 by auto
            qed
            ultimately show ?thesis by auto
          next
            case (MTValue x2)
            then obtain v where vDef:"accessStore locs m' = Some (MValue v)"
              using mcLocs MCon.simps(1)[of x2 m' locs] by (auto split:option.splits
memoryvalue.splits)
            moreover have "accessTypeStore locs m' = Some (MTValue x2)" using stD MTValue by auto
            moreover have "tp = MTValue x2" using calculation in1 by simp
            moreover have "MCon tp m' locs" using mcLocs calculation MTValue by blast
            ultimately show ?thesis using MTValue by simp
          qed
        qed
      qed
    next
      have old:"SomeValSomeTyp mem'" using typeSafeDeclNone_hyps(1) unfolding TypeSafe_def by blast
      have non:"∀loc. LSubPrefL2 loc (ShowL_nat (Toploc m)) ⟶ accessStore loc mem' = None"
```

```
        using 18(7) typeSafeDeclNone_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def
        by auto
      have somesomeT:"∀ destl'.
     TypedMemSubPref destl' (ShowL_nat (Toploc m)) (MTArray x t) ⟶
     (∃ t. accessStore destl' m' = Some t) = (∃ tt. accessTypeStore destl' m' = Some tt)"
        using minit_TypeCompChange_somesome[OF mmDef[symmetric]]
        by (metis allocateSameAccess allocateTypeSameAccess m'Def2)
      show "SomeValSomeTyp m'" unfolding SomeValSomeTyp_def
      proof intros
        fix locs
        show "(∃ t. accessStore locs m' = Some t) = (∃ tt. accessTypeStore locs m' = Some tt) "

        proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc m)) (MTArray x t)")
          case True
          then show ?thesis using somesomeT by simp
        next
          case False
          then have "accessStore locs m' = accessStore locs mem'"
            using "18"(7) nonLocChanged by auto
          moreover have "accessTypeStore locs m' = accessTypeStore locs mem'" using False
            by (metis (no_types, lifting) "18"(7) allocateTypeSameAccess ext m'Def2
minitSingleChange_typedPref_typed mmDef prod.inject)
          ultimately show ?thesis using old
            by (simp add: SomeValSomeTyp_def)
        qed
      qed
    qed
  next
    fix x l
    assume *:"(∄ y y'. x = type.Memory y ∨ x = Value y') ∧ (x, l) |∈| fmran (Denvalue e)"
    then obtain ip' where **: "Denvalue e $$ ip' = Some (x, l)" by blast
    then have "ip' ≠ ip " using * eDef unfolding updateEnv.simps by auto
    then have "Denvalue ev' $$ ip' = Some (x, l)" using 18 ** decl_env_not_i by (auto split:if_splits)
    then show "(x, l) |∈| fmran (Denvalue ev')" using fmranI by metis
  next
    fix sckl ptr
    assume *:" accessStore sckl k' = Some ptr ∧ (∄ y y'. ptr = KMemptr y ∨ ptr = KValue y')"
    have "∃ v. accessStore (ShowL_nat (Toploc k)) k' = (Some (KMemptr v))"
      using k'Def unfolding push_def allocate_def updateStore_def accessStore_def by simp
    then have "sckl ≠ (ShowL_nat (Toploc k))" using * by auto
    then have "accessStore sckl k' =accessStore sckl sck'"
      using k'Def 18 unfolding push_def allocate_def updateStore_def accessStore_def by simp
    then show "accessStore sckl sck' = Some ptr " using * by simp
  next
    show "cd' = c" using sameCd by auto
  next
    have "Toploc m' > Toploc m" using topMM m'Def2 unfolding allocate_def by auto
    then show "Toploc mem' ≤ Toploc m'" using 18(7) by simp
  next
    fix locs v
    assume in1:"accessStore locs mem' = Some v "
    then obtain tloc where tlocDef: " (tloc<Toploc mem' ∧ LSubPrefL2 locs (ShowL_nat (tloc)))"
      using 18 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def by blast
    then have "¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∧ locs ≠ (ShowL_nat (Toploc mem'))" using
18(7) LSubPrefL2_def
      by (metis dual_order.refl in1 mLim old.prod.inject option.discI)
    then show "accessStore locs m' = Some v " using in1 nonLocChanged2 18(7) by simp
  next
    fix locs t
    assume in1:"accessTypeStore locs mem' = Some t"
    then obtain v where in2:"accessStore locs mem' = Some v"
      using 18 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    then obtain tloc where tlocDef: " (tloc<Toploc mem' ∧ LSubPrefL2 locs (ShowL_nat (tloc)))"
      using 18 typeSafeDeclNone_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def by blast
```

```
    then have "¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∧ locs ≠ (ShowL_nat (Toploc mem'))" using
18(7) LSubPrefL2_def
      by (metis dual_order.refl in2 mLim old.prod.inject option.discI)
    then show "accessTypeStore locs m' = Some t"
      using in1 nonLocChanged2_Typed 18(7) by simp
  next
    fix locs
    assume in1:" ∃tloc<Toploc mem'. LSubPrefL2 locs (ShowL_nat (tloc)) ∧ accessStore locs mem' = None"
    then obtain tloc where "tloc<Toploc mem' ∧ LSubPrefL2 locs (ShowL_nat (tloc))" by blast
    then have "¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∧ locs ≠ (ShowL_nat (Toploc mem'))" using
18(7) LSubPrefL2_def
      by (metis MemLSubPrefTransitive Read_Show_nat'_id ShowLNatDot hash_contains_dot nat_neq_iff)
    then show "accessStore locs m' = None" using in1 nonLocChanged2 18(7) by auto

  qed
qed


lemma typeSafeDeclNone:
  shows "declNonePost ev' sck' cd' mem' st e c m' k'" using typeSafeDeclNone_hyps(2)
proof(cases rule:decl.elims)
  case (1 t _ _ _ _ c' m k e')
  from 1 show ?thesis by (rule typeSafeDeclNone_1)
next
  case (2 t v t' _ _ _ _ c m k e)
  then show ?thesis by simp
next
  case 3
  then show ?thesis by simp
next
  case 4
  then show ?thesis by simp
next
  case 5
  then show ?thesis by simp
next
  case 6
  then show ?thesis by simp
next
  case 7
  then show ?thesis by simp
next
  case 8
  then show ?thesis by simp
next
  case (9 x t p t' cd _ _ c m k e)
  then show ?thesis by simp
next
  case (10 x t p t' _ mem _ c m k e)
  then show ?thesis by simp
next
  case (11 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case 12
  then show ?thesis by simp
next
  case 13
  then show ?thesis by simp
next
  case (14 v _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case 15
```

```
  then show ?thesis by simp
next
  case 16
  then show ?thesis by simp
next
  case 17
  then show ?thesis by simp
next
  case (18 x t _ _ _ _ c' m k e')
  from 18 show ?thesis by (rule typeSafeDeclNone_18)
next
  case (19 x t p t' _ mem _ c m k e)
  then show ?thesis by simp
next
  case (20 x t p t' _ _ _ c m k e)
  then show ?thesis by simp
next
  case (21 x t p t' _ cd _ _ c m k e)
  then show ?thesis by simp
next
  case (22 x t p x' t' _ _ _ s c m k e)
  then show ?thesis by simp
next
  case (23 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (24 _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case 25
  then show ?thesis by simp
next
  case (26 v _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (27 v _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (28 _ _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case (29 _ _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case (30 _ _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case 31
  then show ?thesis by simp
next
  case 32
  then show ?thesis by simp
next
  case (33 _ v _ _ _ _)
  then show ?thesis by simp
next
  case 34
  then show ?thesis by simp
next
  case (35 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (36 v _ _ _ _ _ _)
  then show ?thesis by simp
```

```
next
  case (37 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case (38 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case (39 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case 40
  then show ?thesis by simp
next
  case 41
  then show ?thesis by simp
next
  case (42 x t p t' _ _ _ _ c m k e)
  then show ?thesis by simp
next
  case (43 t t' p t'' _ _ _ _ c m k e)
  then show ?thesis by simp
next
  case (44 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (45 v _ _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (46 v _ _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (47 v _ _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (48 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case 49
  then show ?thesis by simp
next
  case 50
  then show ?thesis by simp
next
  case 51
  then show ?thesis by simp
next
  case 52
  then show ?thesis by simp
qed



end

definition ncpElementsNoSubPref::"memoryT ⇒ memoryT ⇒ bool"
  where "ncpElementsNoSubPref lm0 lmD = (∀ i l1 t1 l2 t2 loc i2 loc2. Toploc lm0 > i
                                          ∧ Toploc lm0 ≤ i2
                                          ∧ LSubPrefL2 loc (ShowL_nat i)
                                          ∧ LSubPrefL2 loc2 (ShowL_nat i2)
                                          ∧ MCon (MTArray l1 t1) lm0 loc ⟶
                                      ¬TypedMemSubPrefPtrs lmD l2 t2 loc2 loc
                                          ∧ ¬TypedMemSubPrefPtrs lmD l1 t1 loc loc2)"

definition ncpOMemInDMem::"memoryT ⇒ memoryT ⇒ bool"
```

```
  where "ncpOMemInDMem lmO lmD = (∀ i loc. i < Toploc lmO ∧ LSubPrefL2 loc (ShowL_nat i)
                                  ⟶ accessStore loc lmO = accessStore loc lmD ∧ accessTypeStore loc
lmO = accessTypeStore loc lmD)"
```

Every memory reference in `lev0` (the extended environment) that is below the top location of the original memory `mem` already existed in the original environment `lev` and stack `lk`.

```
definition ncpDenvalueLimit::"environment ⇒ environment ⇒ stack ⇒ stack ⇒ memoryT ⇒ bool "
  where "ncpDenvalueLimit lev0 lev lk lst mem = (∀ tp' locs p i. (type.Memory tp', Stackloc locs) |∈|
fmran (Denvalue lev0)
                                        ∧ accessStore locs lk = Some (KMemptr p) ∧ i < Toploc mem ∧
LSubPrefL2 p (ShowL_nat i)⟶
                                          (∃ tp'' loc2 p'.( (type.Memory tp'', Stackloc loc2) |∈|
fmran (Denvalue lev)
                                                                ∧ accessStore loc2 lst = Some
(KMemptr p') ∧
                                                      ((p' = p ∧ tp'' = tp' ) ∨ (∃ len arr. p' ≠ p ∧
tp'' = MTArray len arr ∧ CompMemType mem len arr tp' p' p))))

                    )"

definition ncpNewSelfPoint::"memoryT ⇒ memoryT ⇒ bool"
  where "ncpNewSelfPoint lmO lmD = (∀ i loc loc2. Toploc lmD > i
                                          ∧ Toploc lmO ≤ i
                                          ∧ LSubPrefL2 loc (ShowL_nat i)
                                          ∧ accessStore loc lmD = Some (MPointer loc2)
                                         ⟶ loc = loc2
                                  )"

lemma ncpElementsNoSubPref_sameMem:
  assumes "TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd"
  shows "ncpElementsNoSubPref (Memory st) (Memory st)" unfolding ncpElementsNoSubPref_def
proof intros
  fix i l1 t1 l2 t2 loc i2 loc2
  assume f1:" i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2
            ∧ LSubPrefL2 loc (ShowL_nat i) ∧ LSubPrefL2 loc2 (ShowL_nat i2) ∧ MCon (MTArray l1 t1)
(Memory st) loc"
  then have stringdif:"(ShowL_nat i) ≠ (ShowL_nat i2)"
    by (metis Read_Show_nat'_id linorder_not_less)
  then have notHash1:"∀ lll. hash loc lll ≠ loc2" using f1
    by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general hash_inequality
hash_suffixes_associative)
  have notHash2:"∀ lll. hash loc2 lll ≠ loc" using f1 stringdif
    by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general hash_inequality
hash_suffixes_associative)

  have "¬ TypedMemSubPrefPtrs (Memory st) l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs (Memory st) l1 t1 loc
loc2"
  proof -
    have " TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd" using assms by simp
    then have limits:"((∀ tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None) ∧
              (∀ loc y. accessStore loc (Memory st) = Some y ⟶ (∃ tloc<Toploc (Memory st). LSubPrefL2
loc (ShowL_nat tloc))))"
      unfolding TypeSafe_def lessThanTopLocs_def by blast
    show "¬ TypedMemSubPrefPtrs (Memory st) l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs (Memory st) l1 t1
loc loc2"
    proof
      show "¬ TypedMemSubPrefPtrs (Memory st) l2 t2 loc2 loc"
      proof
        assume f2:"TypedMemSubPrefPtrs (Memory st) l2 t2 loc2 loc"
        then show False
        proof(cases t2)
          case (MTArray x11 x12)
          then obtain i' l where "(i'<l2 ∧ accessStore (hash loc2 (ShowL_nat i')) (Memory st) = Some
```

```
(MPointer l)
                                  ∧ (l = loc ∨ TypedMemSubPrefPtrs (Memory st) x11 x12 l loc))"
          using TypedMemSubPrefPtrs.simps(2)[of "Memory st" l2 x11 x12 loc2 loc] f2 by blast
        then show ?thesis using f1 limits
          by (metis LSubPrefL2_def hash_suffixes_associative not_Some_eq)
      next
        case (MTValue x2)
        then have "(∃i<l2. hash loc2 (ShowL_nat i) = loc)" using TypedMemSubPrefPtrs.simps(1)[of
"Memory st" l2 x2 loc2 loc] f2 by auto
        then show ?thesis using notHash2 by simp
      qed

    qed
  next
    show "¬ TypedMemSubPrefPtrs (Memory st) l1 t1 loc loc2 "
    proof
      assume f2:"TypedMemSubPrefPtrs (Memory st) l1 t1 loc loc2"
      then show False using f1 notHash1
      proof(induction t1 arbitrary:loc l1 i)
        case (MTArray x11 x12)
        then obtain i' l where i'def:"(i'<l1 ∧ accessStore (hash loc (ShowL_nat i')) (Memory st) =
Some (MPointer l)
                                    ∧ (l = loc2 ∨ TypedMemSubPrefPtrs (Memory st) x11 x12 l loc2))"
          using TypedMemSubPrefPtrs.simps(2)[of "Memory st" l1 x11 x12 loc loc2] f2 by blast
        then show ?thesis
        proof(cases "l = loc2")
          case True
          then have "MCon (MTArray l1 (MTArray x11 x12)) (Memory st) loc" using MTArray by blast
          then have " MCon (MTArray x11 x12) (Memory st) loc2 " using MCon.simps(2)[of l1 t1 "Memory
st" loc] i'def MTArray
            using CompTypeRemainsMCon True CompMemType.simps(2) by blast
          then have "∃x i. accessStore l (Memory st) = Some x ∨ accessStore (hash l (ShowL_nat i))
(Memory st) = Some x"
            using i'def MCon_imps_Some[of " (MTArray x11 x12)" "Memory st" loc2] True by simp
          then show ?thesis using limits True f1
            by (metis LSubPrefL2_def Not_Sub_More_Specific not_Some_eq)
        next
          case False
          then have "MCon (MTArray l1 (MTArray x11 x12)) (Memory st) loc" using MTArray by blast
          then have " MCon (MTArray x11 x12) (Memory st) l " using MCon.simps(2)[of l1 t1 "Memory
st" l] i'def MTArray
            using CompTypeRemainsMCon False CompMemType.simps(2) by blast
          then obtain i'' where i''def: "∃x. accessStore l (Memory st) = Some x ∨ accessStore (hash
l (ShowL_nat i'')) (Memory st) = Some x"
            using i'def MCon_imps_Some[of " (MTArray x11 x12)" "Memory st" l] False by blast
          then obtain newLoc where nlocdef:"newLoc < Toploc (Memory st) ∧ LSubPrefL2 l (ShowL_nat
newLoc)
                                          ∨ LSubPrefL2 (hash l (ShowL_nat i'')) (ShowL_nat newLoc)"
            using limits by blast
          then have "TypedMemSubPrefPtrs (Memory st) x11 x12 l loc2" using i'def False by simp
          then show ?thesis using MTArray.IH[of x11 l newLoc] MTArray i'def nlocdef
            by (metis MemLSubPrefL2_specific_imps_general <MCon (MTArray x11 x12) (Memory st) l>
i''def
                lessThanSome_imps_Locs_hash lessThanTopLocs_def limits linorder_not_less option.discI)
        qed
      next
        case (MTValue x2)
        then have "(∃i<l1. hash loc (ShowL_nat i) = loc2)" using TypedMemSubPrefPtrs.simps(1)[of
"Memory st" l2 x2 loc loc2] f2 by auto
        then show ?thesis using MTValue by simp
      qed
    qed
  qed
  qed
```

```
  moreover show "¬ TypedMemSubPrefPtrs (Memory st) l2 t2 loc2 loc" using calculation by blast
  ultimately show "¬ TypedMemSubPrefPtrs (Memory st) l1 t1 loc loc2" by blast
qed


lemma cpm2mrec_somesome:
  "cpm2mrec srcl destl tp ms a = Some v''' ⟶
  (∀destl'. TypedMemSubPref destl' destl tp ⟶ ((∃t. accessStore destl' v''' = Some t)
      ⟷ (∃tt. accessTypeStore destl' v''' = Some tt)))"
proof(induction tp arbitrary:destl a v''' srcl )
  case (MTArray x1 t)
  show ?case
  proof intros
    fix destl'
    assume **:"cpm2mrec srcl destl (MTArray x1 t) ms a = Some v''''"
      and ***:"TypedMemSubPref destl' destl (MTArray x1 t)"
    have a5:"(case accessStore srcl ms of None ⇒ None | Some (MValue literal) ⇒ None
     | Some (MPointer l) ⇒
        let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a in iter' (λi. cpm2mrec (hash
l
            (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x1)
           = Some v''''"
      using ** cpm2mrec.simps(1)[of srcl destl x1 t ms a] by simp
    then obtain l where ldef:" accessStore srcl ms = Some (MPointer l)" by (auto split:option.splits
memoryvalue.splits)
    then have a6:"Some v''' = (let m = updateTypedStore destl (MPointer destl) (MTArray x1 t) a in
iter' (λi. cpm2mrec (hash l
       (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x1)"
      using a5 by auto
    then obtain m where mdef:"updateTypedStore destl (MPointer destl) (MTArray x1 t) a = m" by auto
    then have v''def:"Some v''' = iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t
ms) m x1" using a6 by simp
    have "(∃t. accessStore destl' v''' = Some t) = (∃tt. accessTypeStore destl' v''' = Some tt)"
      using ***
    proof(induction rule: iter'_induct[OF _ _ v''def[symmetric]])
      case (1)
      then show ?case by simp
    next
      case (2 x v'')
      then obtain v'
        where a10:"iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) m x = Some
v'"
          and a20:"(TypedMemSubPref destl' destl (MTArray x t) ⟶
           (∃t. accessStore destl' v' = Some t) = (∃tt. accessTypeStore destl' v' = Some tt))"
          and a30:"cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" by
blast
      then show ?case
      proof(cases "destl' = destl")
        case True


        then show ?thesis using 2
          by (metis accessStore_updateStore hash_inequality minitSelfPointers
minitSingleChange_typedPref)
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
          case True
          then have "destl' ≠ hash destl (ShowL_nat x)"
            by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
          moreover have "∀i<x. ¬ TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
t" using True
```

```
          using TypedMemSubPref_hash_index_disjoint_typed by blast
        ultimately have i1:"∀i<x. accessTypeStore (hash destl' (ShowL_nat i)) v' = accessTypeStore
(hash destl' (ShowL_nat i)) v''"
          using cpm2mrec_SubPrefixes_typedPref_typed[of _ "(hash destl (ShowL_nat x))" t v' v''] a30
          by (meson ShowLNatDot cpm2mrec_SubPrefixes_typedPref_typed f1 hash_injective)
        have i2:"(∃t. accessStore destl' v' = Some t) = (∃tt. accessTypeStore destl' v' = Some tt)"
          using True a20 by blast

        have i5:"∀locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs v' =
accessStore locs v''"
          using a30 cpm2mrec_SubPrefixes by blast
        moreover have i6:"∀l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l (hash destl
(ShowL_nat x))"
          using TypedMemSubPref_index_disjoint by blast
        ultimately have i7:"∀l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l v' =
accessStore l v''" by auto
        have i8:"∀l l'. TypedMemSubPref l destl (MTArray x t) ∧ accessStore l v' = Some (MPointer
l') ⟶ l' = l"
          using cpm2mSelfPointers[OF a10] by blast

        show ?thesis using i2
          using True a30 i6 i7 cpm2mrec_SubPrefixes_both by auto
      next
        case False
        then have b5:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl
(ShowL_nat x)"
          using 2 TypedMemSubPref.simps(2)[of destl' destl "(Suc x)" t] f1
          using less_Suc_eq
          by (metis TypedMemSubPref.simps(2))
        then have "accessTypeStore destl' m = accessTypeStore destl' v'"
          using False a10 cpm2mSingleChange_typedPref_typed by blast
        then have "cpm2mrec (hash l (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" us-
ing a30 by simp
        then have IH1:"(∀destl'.
      TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ⟶
      (∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' = Some tt))"
          using MTArray.IH[of _ "(hash destl (ShowL_nat x))" v' v''] by simp

        have "(∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' = Some tt)"
        proof(cases "t")
          case (MTArray x11 x12)
          then have a:"Some v'' = (case accessStore (hash l (ShowL_nat x)) ms of None ⇒ None | Some
(MValue literal) ⇒ None
    | Some (MPointer l) ⇒
      let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 x12) v'
      in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12
ms) m x11)"
            using a30 cpm2mrec.simps(1)[of "(hash l (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11 x12
ms v']
            by simp
          then obtain ll where lldef:"accessStore (hash l (ShowL_nat x)) ms = Some (MPointer ll)" by
(auto split:option.splits memoryvalue.splits)
          then have a2:"Some v'' = (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer
(hash destl (ShowL_nat x))) (MTArray x11 x12) v'
      in iter' (λi. cpm2mrec (hash ll (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
          (ShowL_nat i)) x12 ms) m x11)" using a by simp
          then obtain m where mdef:"m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash
destl (ShowL_nat x))) (MTArray x11 x12) v'"
            by simp
          then have v''Def:"Some v'' = iter' (λi. cpm2mrec (hash ll (ShowL_nat i)) (hash (hash destl
              (ShowL_nat x)) (ShowL_nat i)) x12 ms) m x11"
            using a2 lldef by presburger
          then have sameAcc:"accessTypeStore (hash destl (ShowL_nat x)) m = accessTypeStore (hash
```

```
destl (ShowL_nat x)) v''"
            using cpm2mSingleChange_typedPref_typed by presburger
          have "¬ TypedMemSubPref (hash destl (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTArray x11
x12)"
            by (metis LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality
typedPrefix_imp_SubPref)
          then have sameAccV:"accessStore (hash destl (ShowL_nat x)) m = accessStore (hash destl
(ShowL_nat x)) v''"
            using cpm2mSingleChange_typedPref v''Def by presburger
          have acc:"accessTypeStore (hash destl (ShowL_nat x)) m = Some (MTArray x11 x12)"
            using mdef unfolding updateTypedStore_def updateTypeStore_def accessTypeStore_def
updateStore_def by auto

          have accV:"accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
            using sameAccV mdef
            unfolding updateTypedStore_def updateTypeStore_def accessTypeStore_def
              updateStore_def accessStore_def
            by auto
          then show ?thesis
          proof(cases "destl' = hash destl (ShowL_nat x)")
            case True
            then have "CompMemType v'' (Suc x) t t destl destl'"
              using MTArray accV by auto
            moreover have "∀i<x11. accessTypeStore (hash destl' (ShowL_nat i)) v'' = Some x12"
              using cpm2m_TypeCompChangeIndexs[OF v''Def[symmetric]] True by auto
            ultimately show ?thesis using MTArray
              using True acc accV sameAcc by auto
          next
            case False
            then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using b5 by auto
            then have "(∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' =
Some tt)"
              using IH1 MTArray by simp
            then show ?thesis using MTArray accV by auto
          qed
        next
          case (MTValue x2)
          then show ?thesis using IH1 b5 by auto
        qed

        then show ?thesis by blast
      qed
    qed
  qed
  then show "(∃t. accessStore destl' v''' = Some t) = (∃tt. accessTypeStore destl' v''' = Some tt)"
by auto
 qed
next
 case (MTValue x)
 show ?case
 proof intros
  fix destl'
  assume **:" cpm2mrec srcl destl (MTValue x) ms a = Some v'''"
  assume ***:"TypedMemSubPref destl' destl (MTValue x)"
  then have mdef:"Some v''' = (case accessStore srcl ms of None ⇒ None | Some (MValue v) ⇒ Some
(updateTypedStore destl (MValue v) (MTValue x) a)
   | Some (MPointer literal) ⇒ None)"
    using ** cpm2mrec.simps(2)[of srcl destl x ms a] by simp
  then obtain v where vdef: "accessStore srcl ms = Some (MValue v)" by (auto split:option.splits
memoryvalue.splits)
  then have v'''def:"Some v''' = Some (updateTypedStore destl (MValue v) (MTValue x) a)" using mdef
by simp
  then have "accessTypeStore destl v''' = Some (MTValue x)" using mdef
```

```
      unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def updateStore_def
accessStore_def by simp
    moreover have "accessStore destl' v''' = Some (MValue v)"
      using v'''def unfolding accessTypeStore_def updateTypedStore_def updateTypeStore_def
updateStore_def accessStore_def
      using "***" by auto
    ultimately show "(∃ t. accessStore destl' v''' = Some t) = (∃ tt. accessTypeStore destl' v''' = Some
tt)"
      using "***" by fastforce
  qed
qed

lemma cpm2m_TypeCompChange_somesome:
  assumes "iter' (λi m''. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms m'') md x1 =
Some v'"
  shows "(∀ destl'. TypedMemSubPref destl' destl (MTArray x1 t) ⟶
         ((∃ t. accessStore destl' v' = Some t) ⟷ (∃ tt. accessTypeStore destl' v' = Some tt)))"
proof(induction rule: iter'_induct[OF _ _ assms(1)])
  case (1)
  then show ?case by simp
next
  case (2 x v'')
  then obtain v'
    where a10: "iter' (λi. cpm2mrec (hash srcl (ShowL_nat i)) (hash destl (ShowL_nat i)) t ms) md x =
Some v'"
      and a20: "(∀ destl'. TypedMemSubPref destl' destl (MTArray x t) ⟶
            ((∃ t. accessStore destl' v' = Some t) ⟷ (∃ tt. accessTypeStore destl' v' = Some tt)))"
      and a30: "cpm2mrec (hash srcl (ShowL_nat x)) (hash destl (ShowL_nat x)) t ms v' = Some v''" by
blast

  show ?case
  proof intros
    fix destl'
    assume a1:"TypedMemSubPref destl' destl (MTArray (Suc x) t)"
    show "((∃ t. accessStore destl' v'' = Some t) ⟷ (∃ tt. accessTypeStore destl' v'' = Some tt))"
    proof(cases "TypedMemSubPref destl' destl (MTArray x t)")
      case True
      then have "destl' ≠ hash destl (ShowL_nat x)" using True
        by (metis ShowLNatDot TypedMemSubPref.simps(2) hash_int_prefix hashesIntSame
not_less_iff_gr_or_eq typedPrefix_imp_SubPref)
      moreover have "¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using True
        using TypedMemSubPref_index_disjoint_typed by blast
      ultimately have i1:"∀ destl'. destl' ≠ hash destl (ShowL_nat x) ∧ ¬ TypedMemSubPref destl'
          (hash destl (ShowL_nat x)) t ⟶
        accessTypeStore destl' v' = accessTypeStore destl' v''"
        using cpm2mrec_SubPrefixes_typedPref_typed[of "(hash srcl (ShowL_nat x))" ] a30 by simp
      have i2:"(∃ t. accessStore destl' v' = Some t) = (∃ tt. accessTypeStore destl' v' = Some tt)"
        using True a20 by blast

      have store_preserved: "∀ locs. ¬ LSubPrefL2 locs (hash destl (ShowL_nat x)) ⟶ accessStore locs
v' = accessStore locs v''"
        using a30 cpm2mrec_SubPrefixes by blast
      moreover have no_prefix_conflict: "∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ ¬ LSubPrefL2 l
(hash destl (ShowL_nat x))"
        using TypedMemSubPref_index_disjoint by blast
      ultimately have store_consistency: "∀ l. TypedMemSubPref l destl (MTArray x t) ⟶ accessStore l
v' = accessStore l v''"
        by auto

      show ?thesis using i2
        using True ‹¬ TypedMemSubPref destl' (hash destl (ShowL_nat x)) t› ‹destl' ≠ hash
          destl (ShowL_nat x)› i1 store_consistency by force

    next
```

```
      case False
      then have a2:"TypedMemSubPref destl' (hash destl (ShowL_nat x)) t ∨ destl' = hash destl (ShowL_nat
x)"
        using a1 unfolding TypedMemSubPref.simps
        using less_SucE by blast
      then show ?thesis
      proof(cases t)
        case (MTArray x11 x12)
        then have a:"Some v'' = (case accessStore (hash srcl (ShowL_nat x)) ms of None ⇒ None | Some
(MValue literal) ⇒ None
     | Some (MPointer l) ⇒
         let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl (ShowL_nat x)))
(MTArray x11 x12) v'
         in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) x12
ms) m x11)"
          using a30 cpm2mrec.simps(1)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x11 x12
ms v'] by auto
        then obtain l where ldef:"accessStore (hash srcl (ShowL_nat x)) ms = Some (MPointer l)" by
(auto split:option.splits memoryvalue.splits)
        then have a2':"Some v''= (let m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash
destl (ShowL_nat x))) (MTArray x11 x12) v'
         in iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash destl (ShowL_nat x))
             (ShowL_nat i)) x12 ms) m x11)" using a by simp
        then obtain m where mdef:"m = updateTypedStore (hash destl (ShowL_nat x)) (MPointer (hash destl
(ShowL_nat x))) (MTArray x11 x12) v'"
          by simp
        then have v''Def:"Some v'' = iter' (λi. cpm2mrec (hash l (ShowL_nat i)) (hash (hash destl
            (ShowL_nat x)) (ShowL_nat i)) x12 ms) m x11"
          using a2' by presburger
        then have sameAcc:"accessTypeStore (hash destl (ShowL_nat x)) m = accessTypeStore (hash destl
(ShowL_nat x)) v''"
          using cpm2mSingleChange_typedPref_typed by presburger
        have "¬ TypedMemSubPref (hash destl (ShowL_nat x)) (hash destl (ShowL_nat x)) (MTArray x11 x12)"
          by (metis LSubPrefL2_def TypedMemSubPref.simps(2) hash_flatten_right hash_inequality
typedPrefix_imp_SubPref)
        then have sameAccV:"accessStore (hash destl (ShowL_nat x)) m = accessStore (hash destl (ShowL_nat
x)) v''"
          using cpm2mSingleChange_typedPref[of l "(hash destl (ShowL_nat x))" x12 ms m x11 v''] v''Def
by simp
        have acc:"accessTypeStore (hash destl (ShowL_nat x)) m = Some (MTArray x11 x12)"
          using mdef unfolding updateTypedStore_def updateTypeStore_def accessTypeStore_def
updateStore_def by auto

        have accV:"accessStore (hash destl (ShowL_nat x)) v'' = Some (MPointer (hash destl (ShowL_nat
x)))"
          using sameAccV mdef
          unfolding updateTypedStore_def updateTypeStore_def accessTypeStore_def
            updateStore_def accessStore_def
          by auto
        then show ?thesis
        proof(cases "destl' = hash destl (ShowL_nat x)")
          case True
          then have "∀i<x11. TypedMemSubPref (hash destl' (ShowL_nat i)) (hash destl (ShowL_nat x))
(MTArray x11 x12)" by auto
          then have "∀i<x11. accessTypeStore (hash (hash destl (ShowL_nat x)) (ShowL_nat i)) v'' = Some
x12"
            using cpm2m_TypeCompChangeIndexs[OF v''Def[symmetric]] by blast
          moreover have "CompMemType v'' (Suc x) (MTArray x11 x12) (MTArray x11 x12) destl destl'"
            using MTArray True accV unfolding CompMemType.simps by blast
          ultimately show ?thesis using acc sameAcc MTArray accV
            using mtypes.simps(5) True by simp
        next
          case False
          then have "TypedMemSubPref destl' (hash destl (ShowL_nat x)) t" using a2 by blast
```

```
        then have "(∃t. accessStore destl' v'' = Some t) = (∃tt. accessTypeStore destl' v'' = Some
tt)"
          using cpm2mrec_somesome[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" t ms v'
v''] a30 by simp
        then show ?thesis using MTArray accV by auto
      qed
    next
      case (MTValue x2)
      then have "destl' = hash destl (ShowL_nat x)" using a2 by simp
      then have v'':"Some v'' = (case accessStore (hash srcl (ShowL_nat x)) ms of None ⇒ None
   | Some (MValue v) ⇒ Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2) v') |
Some (MPointer literal) ⇒ None)"
        using a30 cpm2mrec.simps(2)[of "(hash srcl (ShowL_nat x))" "(hash destl (ShowL_nat x))" x2 ms v'
] MTValue by simp
      then obtain v where "accessStore (hash srcl (ShowL_nat x)) ms = Some (MValue v)" by (auto
split:option.splits memoryvalue.splits)
      then have "Some v'' = Some (updateTypedStore (hash destl (ShowL_nat x)) (MValue v) (MTValue x2)
v')" using v'' by simp
      then show ?thesis unfolding updateTypeStore_def updateTypedStore_def accessTypeStore_def
updateStore_def using MTValue
        using ‹destl' = hash destl (ShowL_nat x)› accessTypeStore_def
        by (metis TypedMemSubPref.simps(1) a30 cpm2mrec_somesome)
    qed
  qed
 qed
qed
```

Postcondition of *decl* with an initial value.

```
abbreviation declPost :: "environment ⇒ 'a state_scheme ⇒ bool ⇒ memoryT ⇒ environment ⇒
calldataT ⇒ memoryT ⇒ stack ⇒ bool"
  where "declPost ev st cp mem' e c m' k' ≡ TypeSafe e (Accounts st) k' m' (Storage st) c ∧
      (¬cp ⟶ (∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs)
                ∧ Toploc (Memory st) ≤ Toploc (m')
                ∧ ncpDenvalueLimit e ev k' (Stack st ) (Memory st)
                ∧ ncpOMemInDMem (Memory st) m'
                ∧ ncpElementsNoSubPref (Memory st) m'
                ∧ ncpNewSelfPoint (Memory st) m'
      ) ∧ Toploc m' ≥ Toploc mem'"

context
  fixes c cd cd' cp e ev ev' ip k' m' mem' sck' st t'' tp v
  assumes typeSafeDecl_hyps: "TypeSafe ev (Accounts st) (Stack st) (Memory st) (Storage st) cd"
      "valueCon ev cd st v t''"
      "TypeSafe ev' (Accounts st) sck' mem' (Storage st ) cd'"
      "decl ip tp (Some (v,t'')) cp cd (Memory st) (Storage st (Address ev)) (cd', mem', sck', ev') =
Some (c, m', k', e)"
      "¬cp ⟶ (∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs) ∧ Toploc (Memory st) ≤
Toploc (mem')
                ∧ ncpDenvalueLimit ev' ev sck' (Stack st) (Memory st)
                ∧ ncpOMemInDMem (Memory st) mem'
                ∧ ncpElementsNoSubPref (Memory st) mem'
                ∧ ncpNewSelfPoint (Memory st) mem'"
      "∀struct.
       t'' = type.Memory struct ⟶
       (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧ accessStore stloc
(Stack st) = Some (KMemptr p) ∧ (tp'' = struct ∧ v = (KMemptr p)
     ∨ (∃len arr. p ≠ (extractValueType v) ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr struct p (extractValueType v))))"
      "∀struct.
       t'' = Calldata struct ⟶
       (∃stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧ accessStore stloc
(Stack st) = Some (KCDptr p) ∧ (tp'' = struct ∧ v = (KCDptr p)
     ∨ (∃len arr. p ≠ (extractValueType v) ∧ tp'' = MTArray len arr ∧ CompMemType cd len arr struct
p (extractValueType v))))"
```

```
      "∀struct. tp = type.Storage struct ⟶ (∀locs tp. SCon tp locs (Storage st (Address ev))
        ⟶ SCon tp locs (Storage st (Address ev'))) ∧ (∃loc tp'' p. (type.Storage tp'', loc) |∈|
fmran
        (Denvalue ev')
                                                ∧(case loc of Stackloc stloc ⇒
                                                    (accessStore stloc sck' =Some (KStoptr
p) ∧ ((tp'' = struct ∧ v = KStoptr p) ∨
                                                    ((extractValueType v) ≠ p ∧
CompStoToType tp'' struct p (extractValueType v))))
                                                | Storeloc stloc ⇒
                                                    (((tp'' = struct ∧ v = KStoptr stloc)
∨
                                                    ((extractValueType v) ≠ stloc ∧
CompStoToType tp'' struct stloc (extractValueType v))))
                                                ))"
```

begin

A value-type variable initialised with a value, converted to the declared type.

```
lemma typeSafeDecl_2:
  assumes "tp = Value t"
    and "Some (v, t'') = Some (KValue vc, Value t')"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem'', sck'', ev'')"
    and "Some (c, m', k', e) = convert t' t vc ≫= (λv'. Some (cdc', mem'', astack_dup ip (Value t)
(KValue v') (sck'', ev'')))"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 2 = assms
  then obtain v' where a10:"(convert t' t vc) = Some v'" using decl.simps(2) by fastforce
  then have a20:"v' = vc" using convertSame by metis
  have "typeCon t' vc" using typeSafeDecl_hyps(2) 2 by simp
  then have a30:"typeCon t vc" using a10 a20 typeSafeConvert by simp
  show ?thesis
  proof(cases "Denvalue ev' $$ ip")
    case (Some a)
    then have same:"(cdc', mem', sck', ev') = (c, m', k', e)" using 2 a10 unfolding astack_dup.simps
by simp
    show ?thesis
    proof intros
      show "TypeSafe e (Accounts st) k' m' (Storage st) c" using typeSafeDecl_hyps(3) same 2(7) by
blast
      show "⋀locs tp. ¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m' locs" using
typeSafeDecl_hyps(5) same 2(7) by simp
      show "¬ cp ⟹ Toploc (Memory st) ≤ Toploc m'" using typeSafeDecl_hyps(5) same 2(7) by simp
      show "¬ cp ⟹ ncpDenvalueLimit e ev k' (Stack st) (Memory st)" using typeSafeDecl_hyps(5) same
2(7) by simp
      show "¬ cp ⟹ ncpOMemInDMem (Memory st) m'" using typeSafeDecl_hyps(5) same 2(7) by simp
      show "¬ cp ⟹ ncpElementsNoSubPref (Memory st) m'" using typeSafeDecl_hyps(5) same 2(7) by
simp
      show "¬ cp ⟹ ncpNewSelfPoint (Memory st) m'" using typeSafeDecl_hyps(5) same 2(7) by simp
      show "Toploc mem' ≤ Toploc m'" using typeSafeDecl_hyps(5) same 2(7) by simp
    qed
  next
    case None
    then have a40:"(Some (cdc', mem', astack ip (Value t) (KValue vc) (sck', ev'))) = Some (c, m', k',
e)" using 2 decl.simps(2) a10 typeSafeDecl_hyps(1) a20 by simp
    then have a50:"k' = push (KValue vc) sck'" using 2 by force

    have a60:"e = (updateEnv ip (Value t) (Stackloc (ShowL_nat (Toploc sck'))) ev')" using a40 a10 by
simp
```

```
    then have a70:"Denvalue e = Denvalue(ev' (| Denvalue := fmupd ip ((Value t),(Stackloc (ShowL_nat
(Toploc sck')))) (Denvalue ev') |))" by simp
    then have a80:"(Denvalue e) $$ ip = Some ((Value t),(Stackloc (ShowL_nat (Toploc sck'))))" by simp

    show ?thesis unfolding TypeSafe_def
    proof(intros)

      show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 2(7) updateEnvUniqueLocs a40 by
fastforce
    next
      have b2: "compPointers sck' (Denvalue ev')" using typeSafeDecl_hyps(3) 2(7) unfolding
TypeSafe_def by auto
      have "Address ev' = Address e"using a70 a60
        by simp
      moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      ultimately show "compPointers k' (Denvalue e)"
        using compPointersNonStackUpd[of sck' ev' e ip "Value t" k' "KValue vc" ] a70 a50
typeSafeDecl_hyps(3) 2(7) b2 by simp
    next
      have *:"safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(3) unfolding
TypeSafe_def using 2(7) by auto
      have **:"Address ev' = Address e"using a70 a60
        by simp
      have ***:"Contract ev' = Contract e" using a70 a60 by simp
      show "safeContract (Accounts st) (Storage st) " using * by auto
    next
      show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
    next
      show "svalueTypes (Svalue e)" using a60 2(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def
by simp
    next
      have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      then show "lessThanTopLocs k'" using stackPushToplocSafe a50 2(7) by metis
    next
      have "lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      then show "lessThanTopLocs c" using stackPushToplocSafe a50 2(7) a40 by simp
    next
      have "lessThanTopLocs mem'" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      then show "lessThanTopLocs m'" using 2(7) a40 by simp
    next
      have "addressFormat (Address ev')" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      then show "addressFormat(Address e)" using 2(7) a40 by auto
    next
      have "addressFormat (Sender ev')" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
      then show "addressFormat (Sender e)" using 2(7) a40 by auto
    next
      show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c " unfolding typeCompat_def
      proof intros
        fix tDen lDen
        assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
        then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
        then have a100:"(Storage st (Address ev')) = (Storage st (Address e))" using
typeSafeDecl_hyps(3) a60 by simp
        have a110:"m' = mem'" and a115:"cdc' = c" using a40 by simp+
        have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 2(7) by simp
        have a130:"∀v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''" using
            lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def typeSafeDecl_hyps(3)
TypeSafe_def 2(7) by auto
        then have a140:"∀x y. ¬((Denvalue ev') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
            sck'))))" using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 2(7) by
fastforce
        have a150: "∀ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip" using a70 a140 fmranI by auto
        then have a160:"∀ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
```

```
            l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip" using a70
            lessThanTopLocs_def fmranI by auto
        show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
        proof(cases lDen)
          case (Stackloc x1)
          then show ?thesis
          proof (cases "x1 = ShowL_nat (Toploc sck')")
            case True
            then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KValue vc)" using a50 by
(simp add:push_def allocate_def updateStore_def accessStore_def)

            then show ?thesis
            proof(cases "accessStore x1 k'")
              case None
              then show ?thesis using True a170 Stackloc by simp
            next
              case some:(Some a)
              then have a180:"a = (KValue vc) " using a170 True by simp
              have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KValue vc)" using a50 by
(simp add:push_def allocate_def updateStore_def accessStore_def)
              then have "ip'' = ip" using a150 True a90 Stackloc by simp
              then have "tDen = Value t" using a90 a80 by simp
              then show ?thesis using Stackloc some a180 a30 a190 True by simp
            qed
          next
            case False

            then have "ip'' ≠ ip" using a160 a90 Stackloc by simp
            then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev')" using a70 Stackloc fmranI a90
False * by fastforce
            then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 2(7) by blast
            then have a190:"accessStore x1 k' = accessStore x1 sck'" using a50 False by (simp
add:push_def allocate_def updateStore_def accessStore_def)
            then show ?thesis
            proof(cases "accessStore x1 k'")
              case None
              then show ?thesis using a180 a190 Stackloc by simp
            next
              case some:(Some a)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
                using a190 some Stackloc a170 * a180 a110 a115 a100 2(7)
                by (cases a; cases tDen; force+)
            qed
          qed
        next
          case (Storeloc x2)
          then have a170:"ip'' ≠ ip" using a70 Storeloc a90 by auto
          then have a180: "(tDen, lDen) |∈| fmran (Denvalue ev')" using a70 Storeloc fmranI a90 by
fastforce
          then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def using
Storeloc a100 2(7) by (cases tDen;force)
        qed
      qed
    next
      show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp

    next
      have a110:"m' = mem'" using a40 by simp
      then show "⋀locs tp. ¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m' locs" using
typeSafeDecl_hyps(5) 2(7) typeSafeDecl_hyps(5) by (simp)
    next
      assume "¬ cp"
      then show "Toploc (Memory st) ≤ Toploc m'" using typeSafeDecl_hyps(5) 2(7) a40 by blast
```

```
    next
      assume notCP:"¬cp"
      show "ncpDenvalueLimit e ev k' (Stack st) (Memory st) "
        unfolding ncpDenvalueLimit_def
      proof intros
        fix tp' locs p i
        assume a120:" (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' =
Some (KMemptr p) ∧ i <
            Toploc (Memory st) ∧ LSubPrefL2 p (ShowL_nat i)"
        have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''" using
            lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def typeSafeDecl_hyps(3)
TypeSafe_def 2(7) by auto
        then have a140:"∀ x y. ¬((Denvalue ev') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
          using lessThanTopLocs_def Read_Show_nat'_id typeSafeDecl_hyps(3) typeSafeAllStacklocsExist
fmranI 2(7) by fastforce
        have a150: "∀ ip''' . Denvalue e $$ ip''' = Some ((Value t),(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip" using a70 a140 fmranI by auto
        then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
            l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip" using a70
            lessThanTopLocs_def fmranI by auto
        show " ∃ tp'' loc2 p'.
            (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
            accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧
                tp'' = tp' ∨ (∃ len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr tp'
                p' p))
            "
        proof -
          obtain ip'' where ip''def:"(Denvalue e) $$ ip'' = Some (type.Memory tp', Stackloc locs)"
using a120 by blast
          then have a170:"ip'' ≠ ip" using a70 by auto
          then have a180: "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev')" using a70 fmranI
ip''def by fastforce
          have a190:"locs ≠ (ShowL_nat (Toploc sck'))" using a160 a170
            by (metis a140 a180 fmlookup_ran_iff snd_eqD)
          then obtain y' where a200:"accessStore locs sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) 2(7) a180 by blast
          then have "accessStore locs k' = accessStore locs sck'" using a50 a120 a190 unfolding
push_def allocate_def updateStore_def accessStore_def by simp
          then show " ∃ tp'' loc2 p'.
            (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
            accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧
                tp'' = tp' ∨ (∃ len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr tp'
                p' p))"
            using typeSafeDecl_hyps(5) 2(7) a120 a40 a60 a120 a180 notCP ncpDenvalueLimit_def
            by (metis)
        qed

      qed
    next
      show "denvalueTypeCorrectness e k' m'"
        unfolding denvalueTypeCorrectness_def
      proof(intros)
        fix t l ptr_loc
        assume *:" (type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"
        then have "(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev')"
          using a40 unfolding astack.simps updateEnv.simps
          by (metis (no_types, lifting) Environment.unfold_congs(5) type.distinct(3) a80 DenvalueChange
fmlookup_ran_iff option.inject
              prod.inject)
        moreover have "accessStore l k' = accessStore l sck'"
```

```
          using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
          by (metis stackvalue.distinct(3) a50 accessStore_def accessStore_non_changed
accessStore_updateStore allocateSameAccess option.inject
              push_def)
        ultimately show "case t of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
          using a40 typeSafeDecl_hyps(3) * unfolding TypeSafe_def denvalueTypeCorrectness_def using
2(7) by auto
      qed
    next
      show "subPrefixStructuralConsistency m'"
        using typeSafeDecl_hyps(3) a40 2(7) unfolding TypeSafe_def subPrefixStructuralConsistency_def
by auto
    next
      show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a40 2(7) by
blast
    next
      assume notCP:"¬cp"
      then show "ncpOMemInDMem (Memory st) m'" using a40 2(7) typeSafeDecl_hyps by simp
    next
      assume notCP:"¬cp"
      then show "ncpElementsNoSubPref (Memory st) m'" using a40 typeSafeDecl_hyps 2 by blast
    next
      assume notCP:"¬cp"
      then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 2 by blast
    next
      show "Toploc mem' ≤ Toploc m' " using a40 by simp
    qed
  qed
qed
```

A calldata array initialised from a calldata pointer with copying: the array is copied into the new calldata.

```
lemma typeSafeDecl_9:
  assumes "tp = Calldata (MTArray x t)"
    and "Some (v, t'') = Some (KCDptr p, tv0)"
    and "cp"
    and "cd = cdc"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem'', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = Calldata (MTArray x t) ∧ 0 < x ∧ Denvalue
        ev'' $$ ip = None then let l = (ShowL_nat (Toploc cdc')); (uub, c') = allocate cdc' in cpm2m p l
x t cdc c'
            ⤜ (λc''. Some (c'', mem'', astack_dup ip (Calldata (MTArray x t)) (KCDptr l) (sck'',
ev''))) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 9 = assms
  then obtain l where a1:"l = ShowL_nat(Toploc cdc')" by simp
  have tv0:"tv0 = Calldata (MTArray x t)" using decl.simps(9) 9(8) by (simp split:if_split_asm)
  then have locationscd':"(∀ tloc loc. Toploc cdc' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)
      ⟶ accessStore loc cdc' = None)" using typeSafeDecl_hyps 9 unfolding TypeSafe_def
lessThanTopLocs_def by simp
  then have b1:"accessStore l cdc' = None" using a1 using Read_Show_nat'_id using LSubPrefL2_def by
auto
  have b2:"x>0" using 9(8) tv0 a1 by (metis not_None_eq)
  then have Mconcd:"MCon (MTArray x t) cdc (extractValueType v)" using typeSafeDecl_hyps(2) 9(2) 9(4)
tv0 by simp
  obtain c' where a2:"∃ dud. (dud, c') = allocate cdc'" by (simp add: allocate_def)
  then have lNotInC':"accessStore l c' = None" using b1 a1 unfolding allocate_def accessStore_def by
simp
  obtain c'' where a3:"cpm2m p l x t cdc c' = Some c''" using 9 a1 a2 typeSafeDecl_hyps
```

```
    by (metis (no_types, lifting) bind.bind_lzero case_prod_conv option.discI option.exhaust)

  have NoneIp:"Denvalue ev'' $$ ip = None" using 9 by (auto split:if_splits)
  then have a4:"Some (c, m', k', e) = Some (c'', mem'', astack_dup ip (Calldata (MTArray x t)) (KCDptr
l) (sck', ev''))"
    using a1 a2 a3 9(7,8) b2 tv0
    by (metis bind.bind_lunit case_prod_conv)
  then have a20:"k' = push (KCDptr l) sck'" using 9 NoneIp tv0 a3 unfolding astack_dup.simps by simp
  have a30:"e = (updateEnv ip (Calldata (MTArray x t)) (Stackloc (ShowL_nat (Toploc sck'))) ev'')" using
a4 NoneIp unfolding astack_dup.simps by simp
  then have a40:"Denvalue e = Denvalue(ev' (| Denvalue := fmupd ip (Calldata (MTArray x t),(Stackloc
(ShowL_nat (Toploc sck')))) (Denvalue ev'') |)" by simp
  then have a50:"(Denvalue e) $$ ip = Some (Calldata (MTArray x t),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp
  show ?thesis unfolding TypeSafe_def
  proof(intros)
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 9(7) astack_dup_is_astack NoneIp
updateEnvUniqueLocs a4 by fastforce
  next
    have b2: "compPointers sck' (Denvalue ev'')" using typeSafeDecl_hyps(3) 9(7) unfolding TypeSafe_def
by auto
    have "Address ev'' = Address e"using a30 by simp
    moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 9(7) by simp
    ultimately show "compPointers k' (Denvalue e)"
      using compPointersNonStackUpd[of sck' ev' e ip " Calldata (MTArray x t)" k' "KCDptr l" ]
        a40 a50 typeSafeDecl_hyps(3) 9(7) b2 a20 by simp

  next
    show "safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using a30 9(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def by
simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 9(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe a20 9(7) by metis
  next
    have "addressFormat (Address ev') " using typeSafeDecl_hyps(3) TypeSafe_def 9(7) by simp
    then show "addressFormat(Address e)" using 9(7) a4 NoneIp by auto
  next
    have "addressFormat (Sender ev')" using typeSafeDecl_hyps(3) TypeSafe_def 9(7) by simp
    then show "addressFormat (Sender e)" using 9(7) a4 NoneIp by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c" unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev'')) = (Storage st (Address e))" using a30 by simp
      have a110:"m' = mem''" using a4 by simp
      have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 9(7) by simp
      then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''" using
lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
      then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
          sck'))))" using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 9(7) by
fastforce
      then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc
sck'))))
          ⟶ ip''' = ip" using a40 lessThanTopLocs_def fmranI by auto
      then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip" using a40 fmranI by
          auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
```

```
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KCDptr l)" using a20 by
(simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = KCDptr l " using a170 True by simp
            have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KCDptr l)" using a20 by (simp
add:push_def allocate_def updateStore_def accessStore_def)
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have a200:"tDen = Calldata (MTArray x t)" using a90 a50 by simp
            then have a210:"t'' = tDen" using 9(2) tv0 by simp
            have a220:"v = KCDptr p" using 9(2) by simp
            have "c = c''" using a4 by simp
            then have "MCon (MTArray x t) c'' l" using a3 unfolding cpm2m_def
              using MCon_cpm2m[of p l t cdc c' x] Mconcd 9(2)
              by (metis a220 lNotInC' extractValueType.simps(2))
            then have "MCon (MTArray x t) c l" using a4 a110 9(7) by blast
            then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 9(2) by simp
          qed
        next
          case False
          then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs c''"
            using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x c''] by
fastforce

          have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc cdc' = None" using locationscd' a2
lNotInC' a1 by auto
          have "∀ locations. accessStore locations cdc' = accessStore locations c'" using a2
accessAllocate[of c' cdc'] by auto
          then have b110:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cdc' = accessStore locs
c''" using b100 a2 unfolding allocate_def by simp

          then have "ip'' ≠ ip" using a160 a90 Stackloc False by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Stackloc fmranI a90 False
* by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 9 by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'" using a20 False by (simp
add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis
            proof(cases a)
              case (KValue x1)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
                using a190 some Stackloc a170 * a180 a110 a100 9(7) by (cases tDen, fastforce+)
            next
              case (KCDptr x2)
              then have b10:"(case tDen of Calldata struct ⇒ MCon struct cdc' x2 | _ ⇒ False)"
                using some KCDptr a170 Stackloc a190 using typeSafeDecl_hyps(3) 9(7) unfolding
TypeSafe_def typeCompat_def by force
              then obtain struct where tden: "tDen = Calldata struct" by (auto split:type.splits)
```

```
            then have "MCon struct cdc' x2" using b10 by simp
            then have "MCon struct c'' x2" using mconCopySingle[of l cdc' c'' struct x2] b110 b105
by auto
            then show ?thesis using Stackloc some KCDptr tden a110 a4 by simp
          next
            case (KMemptr x3)
            then have b10:"(case tDen of type.Memory struct ⇒ MCon struct mem' x3 | _ ⇒ False)"
              using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 9(7) unfolding TypeSafe_def
typeCompat_def by force
            then obtain struct where tden: "tDen = type.Memory struct" by (auto split:type.splits)
            then have "MCon struct mem' x3" using b10 by simp
            then show ?thesis using Stackloc some KMemptr tden a110 a4 9 by simp
          next
            case (KStoptr x4)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a100 9(7) by (cases tDen; fastforce)
          qed
        qed
      qed
    next
      case (Storeloc x2)
      then have "ip'' ≠ ip" using a160 a90 by simp
      then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Storeloc fmranI a90 * by
fastforce
      then have a180:"(case tDen of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev'')) | _
⇒ False)" using a170 Storeloc using typeSafeDecl_hyps(3) 9(7)
        unfolding TypeSafe_def typeCompat_def by force
      then obtain typ' where tdent:"tDen = type.Storage typ'" by (auto split: type.splits)
      then have "SCon typ' x2 (Storage st (Address ev''))" using a180 by simp
      then have "SCon typ' x2 (Storage st (Address e))" using a100 by simp
      then show ?thesis using a170 Storeloc using typeSafeDecl_hyps(3) 9(7) tdent
        by simp
    qed
  qed
next
  have b100:"lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) 9(7) unfolding TypeSafe_def by blast
  have b99:"Toploc c' = Toploc c" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
cdc c' x c''] b2 by simp
  have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs c"
    using cpm2mSingleChange[of p l t cdc c' x] a4 a3 cpm2m_def[of p l x t cdc c' ] by fastforce
  moreover have "∀locations. accessStore locations cdc' = accessStore locations c'" using a2
accessAllocate[of c' cdc'] by auto
  ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cdc' = accessStore locs c"
by simp
  then show "lessThanTopLocs c" using cdMemLocsToploc[of cdc' l c' c] a1 a2 b100 b99 by auto
next
  show "lessThanTopLocs m'" using typeSafeDecl_hyps(3) 9(7) a4 unfolding TypeSafe_def by auto

next
  show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
next
  have a110:"m' = mem'" using a4 9 by simp
  then show "⋀locs tp. ¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m' locs" using
typeSafeDecl_hyps(5) 9(7) typeSafeDecl_hyps(5) by (simp)
next
  show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
  proof(intros)
    fix t l ptr_loc
    assume *:" (type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some (KMemptr
ptr_loc)"
    then have "(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev')"
      using a4 unfolding astack.simps updateEnv.simps
      by (metis (no_types, lifting) type.distinct(7) a50 typeSafeDecl_hyps(4) decl_env_not_i
fmlookup_ran_iff
```

```
          option.inject prod.inject)
      moreover have "accessStore l k' = accessStore l sck'"
        using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
        by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
          fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
      ultimately show "case t of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
        using a4 typeSafeDecl_hyps(3) * unfolding TypeSafe_def denvalueTypeCorrectness_def using 9(7)
by auto
    qed
  next
    show "subPrefixStructuralConsistency m'"
      using typeSafeDecl_hyps(3) a4 9(7) unfolding TypeSafe_def subPrefixStructuralConsistency_def by
auto
  next
    show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a4 9(7) by blast
  next
    assume notCp:"¬cp"
    then show "Toploc (Memory st) ≤ Toploc m'" using a4 typeSafeDecl_hyps 9 by auto
    then show "ncpDenvalueLimit e ev k' (Stack st) (Memory st) " using 9(3) notCp by simp
    show "ncpOMemInDMem (Memory st) m'" using a4 typeSafeDecl_hyps 9 notCp by blast
    show "ncpElementsNoSubPref (Memory st) m'"
      using a4 typeSafeDecl_hyps 9 notCp by blast
  next
    assume notCP:"¬cp"
    then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 9 by blast
  next
    show "Toploc mem' ≤ Toploc m' " using 9 typeSafeDecl_hyps a4 by blast
  qed
qed
```

A calldata array initialised from a memory pointer with copying: the array is copied into the new calldata.

```
lemma typeSafeDecl_10:
  assumes "tp = Calldata (MTArray x t)"
    and "Some (v, t'') = Some (KMemptr p, tv0)"
    and "cp"
    and "cd = cd0"
    and "state.Memory st = mem"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem''', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = type.Memory (MTArray x t) ∧ 0 < x ∧ Denvalue
        ev'' $$ ip = None then let l = (ShowL_nat (Toploc cdc')); (uub, c') = allocate cdc' in cpm2m p l
x t mem c'
            ≫= (λc''. Some (c'', mem''', astack_dup ip (Calldata (MTArray x t)) (KCDptr l) (sck'',
ev''))) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 10 = assms
  then obtain l where a1:"l = ShowL_nat(Toploc cdc')" by simp
  have vk:"tv0 = type.Memory (MTArray x t)" using decl.simps(9) 10(8) by (simp split:if_split_asm)
  then have locationscd':"(∀ tloc loc. Toploc cdc' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)
      ⟶ accessStore loc cdc' = None)" using typeSafeDecl_hyps 10 unfolding TypeSafe_def
lessThanTopLocs_def by simp
  then have b1:"accessStore l cdc' = None" using a1 using Read_Show_nat'_id using LSubPrefL2_def by
auto
  have b2:"x>0" using 10(8) vk a1 by (metis not_None_eq)
  then have Mconcd:"MCon (MTArray x t) mem (extractValueType v)" using typeSafeDecl_hyps(2) 10(2)
10(4) vk 10(5) by simp
  obtain c' where a2:"∃ dud. (dud, c') = allocate cdc'" by (simp add: allocate_def)
  then have lNotInC':"accessStore l c' = None" using b1 a1 unfolding allocate_def accessStore_def by
simp
```

obtain *c''* **where** *a3:"cpm2m p l x t mem c' = Some c''"* **using** *10 a1 a2 typeSafeDecl_hyps* **by** *(metis (no_types, lifting) bind.bind_lzero case_prod_conv option.discI option.exhaust)*
have *NoneIp:"Denvalue ev'' $$ ip = None"* **using** *10* **by** *(auto split:if_splits)*
then have *a4:"Some (c, m', k', e) = Some (c'', mem''', astack_dup ip (Calldata (MTArray x t)) (KCDptr l) (sck'', ev''))"*
using *a1 a2 a3 10 typeSafeDecl_hyps decl.simps(9) vk*
by *(metis (no_types, lifting) b2 bind.bind_lunit case_prod_conv)*
then have *a4:"Some (c, m', k', e) = Some (c'', mem''', astack_dup ip (Calldata (MTArray x t)) (KCDptr l) (sck', ev''))"*
using *10(7)* **by** *blast*
then have *a20:"k' = push (KCDptr l) sck'"* **using** *10 NoneIp* **by** *force*
have *a30:"e = (updateEnv ip (Calldata (MTArray x t)) (Stackloc (ShowL$_{nat}$ (Toploc sck'))) ev'')"* **using** *a4 NoneIp* **by** *simp*
then have *a40:"Denvalue e = Denvalue(ev' (| Denvalue := fmupd ip (Calldata (MTArray x t),(Stackloc (ShowL$_{nat}$ (Toploc sck')))) (Denvalue ev'') |))"* **by** *simp*
then have *a50:"(Denvalue e) $$ ip = Some (Calldata (MTArray x t),(Stackloc (ShowL$_{nat}$ (Toploc sck'))))"* **by** *simp*
show *?thesis* **unfolding** *TypeSafe_def*
proof*(intros)*
show *"unique_locations (Denvalue e)"* **using** *typeSafeDecl_hyps(3) 10(7) updateEnvUniqueLocs a4 NoneIp* **by** *fastforce*
next
have *b2: "compPointers sck' (Denvalue ev'')"* **using** *typeSafeDecl_hyps(3) 10(7)* **unfolding** *TypeSafe_def* **by** *auto*
have *"Address ev'' = Address e"***using** *a30* **by** *simp*
moreover have *"lessThanTopLocs sck'"* **using** *typeSafeDecl_hyps(3) TypeSafe_def 10(7)* **by** *simp*
ultimately show *"compPointers k' (Denvalue e)"*
using *compPointersNonStackUpd[of sck' ev' e ip " Calldata (MTArray x t)" k' "KCDptr l"] a40 a50 typeSafeDecl_hyps(3) 10(7) b2 a20* **by** *simp*

next
show *"safeContract (Accounts st) (Storage st)"* **using** *typeSafeDecl_hyps(1) TypeSafe_def* **by** *simp*
next
show *"balanceTypes (Accounts st)"* **using** *typeSafeDecl_hyps(1) TypeSafe_def* **by** *simp*
next
show *"svalueTypes (Svalue e)"* **using** *a30 10(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def* **by** *simp*
next
have *"addressFormat (Address ev') "* **using** *typeSafeDecl_hyps(3) TypeSafe_def 10(7)* **by** *simp*
then show *"addressFormat (Address e)"* **using** *10(7) a4 NoneIp* **by** *auto*
next
have *"addressFormat (Sender ev')"* **using** *typeSafeDecl_hyps(3) TypeSafe_def 10(7)* **by** *simp*
then show *"addressFormat (Sender e)"* **using** *10(7) a4 NoneIp* **by** *auto*
next
have *"lessThanTopLocs sck'"* **using** *typeSafeDecl_hyps(3) TypeSafe_def 10(7)* **by** *simp*
then show *"lessThanTopLocs k'"* **using** *stackPushToplocSafe a20 10(7)* **by** *metis*
next
show *"typeCompat (Denvalue e) k' m' (Storage st (Address e)) c"* **unfolding** *typeCompat_def*
proof *intros*
fix *tDen lDen*
assume **: "(tDen, lDen) |∈| fmran (Denvalue e)"*
then obtain *ip''* **where** *a90:"Denvalue e $$ ip'' = Some (tDen, lDen)"* **using** *** **by** *auto*
then have *a100:"(Storage st (Address ev'')) = (Storage st (Address e))"* **using** *a30* **by** *simp*
have *a110:"m' = mem'''"* **using** *a4* **by** *simp*
have *a120:"lessThanTopLocs sck'"* **using** *typeSafeDecl_hyps(3) TypeSafe_def 10(7)* **by** *simp*
then have *a130:"∀ v'''. accessStore (ShowL$_{nat}$ (Toploc sck')) sck' ≠ Some v'''"* **using** *lessThanTopLocs_def Read_Show_nat'_id* **using** *LSubPrefL2_def* **by** *auto*
then have *a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL$_{nat}$ (Toploc sck'))))"* **using** *TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 10(7)* **by** *fastforce*
then have *a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL$_{nat}$ (Toploc sck'))))*
⟶ ip''' = ip" **using** *a40 lessThanTopLocs_def fmranI* **by** *auto*
then have *a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧*

```
        l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip" using a40 fmranI by
        auto
    show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
    proof(cases lDen)
      case (Stackloc x1)
      then show ?thesis
      proof (cases "x1 = ShowL_nat (Toploc sck')")
        case True
        then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KCDptr l)"
          using a20 by (simp add:push_def allocate_def updateStore_def accessStore_def)
        then show ?thesis
        proof(cases "accessStore x1 k'")
          case None
          then show ?thesis using True a170 Stackloc by simp
        next
          case some:(Some a)
          then have a180:"a = KCDptr l " using a170 True by simp
          have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KCDptr l)"
            using a20 by (simp add:push_def allocate_def updateStore_def accessStore_def)
          then have "ip'' = ip" using a150 True a90 Stackloc by simp
          then have a200:"tDen = Calldata (MTArray x t)" using a90 a50 by simp
          have a220:"v = KMemptr p" using 10(2) by simp
          have "c = c''" using a4 by simp
          then have "MCon (MTArray x t) c'' l" using a3 unfolding cpm2m_def
            using MCon_cpm2m[of p l t mem c' x c'' ] Mconcd 10(2)
            by (metis a220 lNotInC' extractValueType.simps(3))
          then have "MCon (MTArray x t) c l" using a4 a110 10(7) by blast
          then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 10(2) by simp
        qed
      next
        case False
        then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs c''"
          using a3 a1 cpm2m_def[of p l x t mem c'] cpm2mSingleChange[of p l t mem c' x c''] by
fastforce

        have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc cdc' = None" using locationscd' a2
lNotInC' a1 by auto
        have "∀ locations. accessStore locations cdc' = accessStore locations c'" using a2
accessAllocate[of c' cdc'] by auto
        then have b110:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cdc' = accessStore locs
c''"
          using b100 a2 unfolding allocate_def by simp

        then have "ip'' ≠ ip" using a160 a90 Stackloc False by simp
        then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')"
          using a40 Stackloc fmranI a90 False * by fastforce
        then obtain y' where a180:"accessStore x1 sck' = Some y' "
          using typeSafeAllStacklocsExist typeSafeDecl_hyps(3) Stackloc 10 by blast
        then have a190:"accessStore x1 k' = accessStore x1 sck'"
          using a20 False by (simp add:push_def allocate_def updateStore_def accessStore_def)
        then show ?thesis
        proof(cases "accessStore x1 k'")
          case None
          then show ?thesis using a180 a190 Stackloc by simp
        next
          case some:(Some a)
          then show ?thesis
          proof(cases a)
            case (KValue x1)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a100 10(7) by (cases tDen, fastforce+)
          next
            case (KCDptr x2)
            then have b10:"(case tDen of Calldata struct ⇒ MCon struct cdc' x2 | _ ⇒ False)"
```

```
              using some KCDptr a170 Stackloc a190 using typeSafeDecl_hyps(3) 10(7) unfolding
TypeSafe_def typeCompat_def by force
            then obtain struct where tden: "tDen = Calldata struct" by (auto split:type.splits)
            then have "MCon struct cdc' x2" using b10 by simp
            then have "MCon struct c'' x2" using mconCopySingle[of l cdc' c'' struct] b110 b105 by
auto
            then show ?thesis using Stackloc some KCDptr tden a110 a4 by simp
          next
            case (KMemptr x3)
            then have b10:"(case tDen of type.Memory struct ⇒ MCon struct mem' x3 | _ ⇒ False)"
              using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 10(7) unfolding TypeSafe_def
typeCompat_def by force
            then obtain struct where tden: "tDen = type.Memory struct" by (auto split:type.splits)
            then have "MCon struct mem''' x3" using b10 10 by simp
            then show ?thesis using Stackloc some KMemptr tden a110 a4 by simp
          next
            case (KStoptr x4)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a100 10(7) by (cases tDen; fastforce)
          qed
        qed
      qed
    next
      case (Storeloc x2)
      then have "ip'' ≠ ip" using a160 a90 by simp
      then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Storeloc fmranI a90 * by
fastforce
      then have a180:"(case tDen of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev'')) | _
⇒ False)"
        using a170 Storeloc using typeSafeDecl_hyps(3) 10(7)
        unfolding TypeSafe_def typeCompat_def by force
      then obtain typ' where tdent:"tDen = type.Storage typ'" by (auto split: type.splits)
      then have "SCon typ' x2 (Storage st (Address ev''))" using a180 by simp
      then have "SCon typ' x2 (Storage st (Address e))" using a100 by simp
      then show ?thesis using a170 Storeloc using typeSafeDecl_hyps(3) 10(7) tdent
        by simp
    qed
  qed
next
  have b100:"lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) 10(7) unfolding TypeSafe_def by blast
  have b99:"Toploc c' = Toploc c" using a3 a4 cpm2m_def[of p l x t mem c' ] cpm2mTopLocSame[of p l t
mem c' x c''] b2 by simp
  have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs c"
    using cpm2mSingleChange[of p l t mem c' x] a4 a3 cpm2m_def[of p l x t mem c' ] by fastforce
  moreover have "∀locations. accessStore locations cdc' = accessStore locations c'" using a2
accessAllocate[of c' cdc'] by auto
  ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs cdc' = accessStore locs c"
by simp
  then show "lessThanTopLocs c" using cdMemLocsToploc[of cdc' l c' c] a1 a2 b100 b99 by auto
next
  show "lessThanTopLocs m'" using typeSafeDecl_hyps(3) 10(7) a4 unfolding TypeSafe_def by auto
next
  show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
next
  show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
  proof(intros)
    fix t l ptr_loc
    assume *:" (type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some (KMemptr
ptr_loc)"
    then have "(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev')"
      using a4 unfolding astack.simps updateEnv.simps
      by (metis (no_types, lifting) type.distinct(7) a50 typeSafeDecl_hyps(4) decl_env_not_i
fmlookup_ran_iff
          option.inject prod.inject)
```

```
      moreover have "accessStore l k' = accessStore l sck'"
        using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
        by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
            fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
      ultimately show "case t of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
        using a4 typeSafeDecl_hyps(3) * unfolding TypeSafe_def denvalueTypeCorrectness_def using 10(7)
by auto
    qed
  next
    show "subPrefixStructuralConsistency m'"
      using typeSafeDecl_hyps(3) a4 10(7) unfolding TypeSafe_def subPrefixStructuralConsistency_def by
auto
  next
    assume notCp:"¬cp"
    then show "Toploc (Memory st) ≤ Toploc m'" using a4 typeSafeDecl_hyps 10 by auto
    then show "ncpDenvalueLimit e ev k' (Stack st) (Memory st) " using 10(3) notCp by simp
    show "ncpOMemInDMem (Memory st) m'" using a4 typeSafeDecl_hyps 10 notCp by blast
    show "ncpElementsNoSubPref (Memory st) m'"
      using a4 typeSafeDecl_hyps 10 notCp by blast
  next
    show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a4 10(7) by
blast
  next
    fix locs tp assume "¬cp"
    then show "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using a4 typeSafeDecl_hyps 10 by blast
  next
    assume notCP:"¬cp"
    then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 10 by blast
  next
    show "Toploc mem' ≤ Toploc m' " using 10 typeSafeDecl_hyps a4 by blast
  qed
qed
```

A memory array initialised from a memory pointer with copying: the array is copied into fresh memory.

```
lemma typeSafeDecl_19:
  assumes "tp = type.Memory (MTArray x t)"
    and "Some (v, t'') = Some (KMemptr p, tv0)"
    and "cp"
    and "cd = cd0"
    and "state.Memory st = mem"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem''', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = type.Memory (MTArray x t) ∧ 0 < x ∧ Denvalue
        ev'' $$ ip = None then cpm2m p (ShowL_nat (Toploc mem''')) x t mem (snd (allocate
        mem''')) ≫= (λm'. Some (cdc', m', astack_dup ip (type.Memory (MTArray x t)) (KMemptr (ShowL_nat
(Toploc mem''')))) (sck'', ev''))) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 19 = assms
  then obtain l where a1:"l = ShowL_nat(Toploc mem')" by simp
  have vk:"tv0 = type.Memory (MTArray x t)" using decl.simps(9) 19(8) by (simp split:if_split_asm)
  then have locationscd':"(∀ tloc loc. Toploc mem' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc mem' = None)"
    using typeSafeDecl_hyps 19 unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have b1:"accessStore l mem' = None" using a1 using Read_Show_nat'_id using LSubPrefL2_def by
auto
  have b2:"x>0" using 19(8) vk a1 by (metis not_None_eq)
  then have Mconcd:"MCon (MTArray x t) mem (extractValueType v)" using typeSafeDecl_hyps(2) 19(2)
19(4) vk 19(5) by simp
  obtain c' where a2:"∃ dud. (dud, c') = allocate mem'" by (simp add: allocate_def)
```

```
  then have aloc:"snd (allocate mem') = c'" by (simp add: allocate_def)
  then have lNotInC':"accessStore l c' = None" using b1 a1 unfolding allocate_def accessStore_def by
auto
  have NoneIp:"Denvalue ev'' $$ ip = None" using 19 by (simp split:if_splits)
  then obtain mm' where a3:"cpm2m p l x t mem c' = Some mm'" using 19 a1 a2 typeSafeDecl_hyps
    using b2 vk aloc by fastforce

  have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2 lNotInC' a1
by auto
  have "∀locations. accessStore locations c' = accessStore locations mem'" using a2 accessAllocate[of
c' mem'] by auto
  then have b27:"∀locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStore locs mem' = accessStore locs
mm'"
    using a3 a1 cpm2m_def[of p l x t mem c'] cpm2mSingleChange[of p l t mem c' x mm'] a2 unfolding
allocate_def by fastforce

  then have subMapping:"Mapping mem' ⊆_f Mapping mm'" using b105 unfolding fmsubset_alt_def
    by (metis (mono_tags, lifting) accessStore_def fmpredI option.discI)
  then have a4:"Some (c, m', k', e) = Some (cdc', mm', astack_dup ip (type.Memory (MTArray x t))
(KMemptr l) (sck', ev'))"
    using a1 a2 a3 19 typeSafeDecl_hyps decl.simps(9) vk b2 aloc NoneIp
    by simp

  then have a20:"k' = push (KMemptr l) sck'" using NoneIp 19 unfolding astack_dup.simps by simp
  have b3:"Address ev' = Address e" using a4 NoneIp 19(7) by simp

  have a30:"e = (updateEnv ip (type.Memory (MTArray x t)) (Stackloc (ShowL_nat (Toploc sck'))) ev')"
using a4 NoneIp 19(7) by simp
  then have a40:"Denvalue e = Denvalue(ev' (| Denvalue := fmupd ip (type.Memory (MTArray x t),(Stackloc
(ShowL_nat (Toploc sck')))) (Denvalue ev') |))" by simp
  then have a50:"(Denvalue e) $$ ip = Some (type.Memory (MTArray x t),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp
  have selfPoint:"∀la l'. TypedMemSubPref la l (MTArray x t) ∧ accessStore la mm' = Some (MPointer l')
⟶ l' = la"
    using cpm2mSelfPointers[of p l t mem c' x mm'] a3 a1 cpm2m_def[of p l x t mem c'] by argo
  have a110:"m' = mm'" using a4 by simp

  then have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                        ⟶ accessTypeStore locs c' = accessTypeStore locs m'"
    using cpm2mSingleChange_Typed[of ] allocateTypeSameAccess a3 a1 unfolding cpm2m_def by blast
  moreover have "∀locs. accessTypeStore locs c' = accessTypeStore locs mem'" using aloc
allocateTypeSameAccess by metis
  ultimately have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                        ⟶ accessTypeStore locs mem' = accessTypeStore locs m'" by
simp
  have "∀destl'.
       TypedMemSubPref destl' l (MTArray x t) ⟶
       (∃st. CompMemType mm' x t st l destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) mm' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' mm' = Some (MTValue pval)))"
    using a1 a3 unfolding cpm2m_def using cpm2m_TypeCompChange[of p l t mem c' x mm'] by blast
  then have mtp:"∀destl'.
       TypedMemSubPref destl' l (MTArray x t) ⟶
       (∃st. CompMemType m' x t st l destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval)))" using a110 by auto

  show ?thesis unfolding TypeSafe_def
  proof(intros)
    fix locs tp assume "¬cp"
```

```
    then show "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using 19 typeSafeDecl_hyps(5) by (simp
split:if_splits)
  next
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 19(7) updateEnvUniqueLocs a4
NoneIp by fastforce
  next
    have b2: "compPointers sck' (Denvalue ev')" using typeSafeDecl_hyps(3) 19(7) unfolding TypeSafe_def
by auto
    moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 19(7) by simp
    ultimately show "compPointers k' (Denvalue e)"
      using compPointersNonStackUpd[of sck' ev' e ip "type.Memory (MTArray x t)" k' "KMemptr l" ]
        a40 a50 typeSafeDecl_hyps(3) 19(7) b2 a20 b3 by simp
  next
    show "safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using a30 19(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def by
simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 19(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe a20 19(7) by metis
  next
    have "addressFormat (Sender ev')" using typeSafeDecl_hyps(3) TypeSafe_def 19(7) by simp
    then show "addressFormat (Sender e)" using 19(7) a4 NoneIp 19(7) by auto
  next
    have "addressFormat (Address ev')" using typeSafeDecl_hyps(3) TypeSafe_def 19(7) by simp
    then show "addressFormat(Address e) " using 19(7) a4 NoneIp 19(7) by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c" unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev')) = (Storage st (Address e))" using a30 by simp
      have a110:"m' = mm'" using a4 by simp
      have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 19(7) by simp
      then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
        using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
      then have a140:"∀ x y. ¬((Denvalue ev') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
        using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 19(7) by fastforce
      then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc
sck')))) ⟶ ip''' = ip"
        using a40 lessThanTopLocs_def fmranI by auto
      then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''')
                        ∧ l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using a40 lessThanTopLocs_def fmranI by auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by
(simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = KMemptr l " using a170 True by simp
            have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by (simp
```

```
add:push_def allocate_def updateStore_def accessStore_def)
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have a200:"tDen = type.Memory (MTArray x t)" using a90 a50 by simp
            have a220:"v = KMemptr p" using 19(2) by simp
            have "c = cdc'" using a4 by simp
            then have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
              using MCon_cpm2m[of p l t mem c' x mm' ] Mconcd 19(2) b2
              by (metis a220 lNotInC' extractValueType.simps(3))
            then have "MCon (MTArray x t) m' l" using a4 a110 19(7) by blast
            then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 19(2) by simp
          qed
        next
          case False
          then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
            using a3 a1 cpm2m_def[of p l x t mem c'] cpm2mSingleChange[of p l t mem c' x mm'] by
fastforce

          have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2
lNotInC' a1 by auto
          have "∀ locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
          then have b110:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs
mm'"
            using b100 a2 unfolding allocate_def by simp

          then have "ip'' ≠ ip" using a160 a90 Stackloc False by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev')" using a40 Stackloc fmranI a90 False
* by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 19 by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'"
            using a20 False by (simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis
            proof(cases a)
              case (KValue x1)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
                using a190 some Stackloc a170 * a180 a110 a100 19(7) by (cases tDen; fastforce)
            next
              case (KCDptr x2)
              then have b10:"(case tDen of Calldata struct ⇒ MCon struct cdc' x2 | _ ⇒ False)"
                using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 19(7) unfolding TypeSafe_def
typeCompat_def by force
              then obtain struct where tden: "tDen = Calldata struct" by (auto split:type.splits)
              then have "MCon struct cdc' x2" using b10 by simp
              then show ?thesis using Stackloc some KCDptr tden a110 a4 by simp
            next
              case (KMemptr x3)
              then have b10:"(case tDen of type.Memory struct ⇒ MCon struct mem' x3 | _ ⇒ False)"
                using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 19(7) unfolding TypeSafe_def
typeCompat_def by force
              then obtain struct where tden: "tDen = type.Memory struct" by (auto split:type.splits)
              then have "MCon struct mem' x3" using b10 by simp
              then have "MCon struct mm' x3" using mconCopySingle[of l mem' mm' struct] b110 b105 by
auto
              then show ?thesis using Stackloc some KMemptr tden a110 a4 by simp
            next
              case (KStoptr x4)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
```

```
            using a190 some Stackloc a170 * a180 a110 a100 19(7) by (cases tDen; fastforce)
          qed
        qed
      qed
    next
      case (Storeloc x2)
      then have "ip'' ≠ ip" using a160 a90 by simp
      then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev')" using a40 Storeloc fmranI a90 * by
fastforce
      then have a180:"(case tDen of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev')) | _ ⇒
False)"
        using a170 Storeloc using typeSafeDecl_hyps(3) 19(7)
        unfolding TypeSafe_def typeCompat_def by force
      then obtain typ' where tdent:"tDen = type.Storage typ'" by (auto split: type.splits)
      then have "SCon typ' x2 (Storage st (Address ev'))" using a180 by simp
      then have "SCon typ' x2 (Storage st (Address e))" using a100 by simp
      then show ?thesis using a170 Storeloc using typeSafeDecl_hyps(3) 19(7) tdent
        by simp
    qed
  qed
next
  show "lessThanTopLocs c" using typeSafeDecl_hyps(3) 19(7) a4 unfolding TypeSafe_def by auto
next
  have b100:"lessThanTopLocs mem'" using typeSafeDecl_hyps(3) 19(7) unfolding TypeSafe_def by blast
  have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t mem c' ] cpm2mTopLocSame[of p l t
mem c' x mm'] b2 by simp
  have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs m'"
    using cpm2mSingleChange[of p l t mem c' x mm'] a4 a3 cpm2m_def[of p l x t mem c'] by fastforce
  moreover have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
  ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs m' = accessStore locs mem'"
by simp
  show "lessThanTopLocs m'" using cdMemLocsToploc[of mem' l c' m'] b100 a1 a2 b99 using b10 by
presburger
next
  show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
next
  have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t mem c' ] cpm2mTopLocSame[of p l t
mem c' x mm'] b2 by simp

  show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
  proof(intros)
    fix t2 l ptr_loc
    assume *:" (type.Memory t2, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"

    show "case t2 of
     MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
    proof(cases "(type.Memory t2, Stackloc l) |∈| fmran(Denvalue ev')")
      case True
      moreover have sameACC:"accessStore l k' = accessStore l sck'"
        using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
        by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
            fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
      then have old:"(case t2 of
                      MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' =
Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc mem' = Some (MTValue val))"
        using typeSafeDecl_hyps(1) 19(7) * True TypeSafe_def typeSafeDecl_hyps(3)
denvalueTypeCorrectness_def by fastforce
      have mcOld:"MCon t2 mem' ptr_loc" using * sameACC True typeSafeDecl_hyps(1) 19(7)
        unfolding TypeSafe_def typeCompat_def
```

```
          by (metis typeSafeDecl_hyps(3) sameMemTSafe)

        have lims:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) ptr_loc"
          using typeSafeAllPtrsNotTop[OF typeSafeDecl_hyps(3) True] * 19(7) sameACC b99 aloc unfolding
allocate_def
          using LSubPrefL2_def MemLSubPrefTransitive by force
        then show ?thesis
        proof(cases t2)
          case (MTArray x11 x12)
          have nonLocChanged2_Typed_cond:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶
accessTypeStore loc m' = accessTypeStore loc mem'"
            using nonLocChanged2_Typed by simp
          have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' = Some x12)" using old
MTArray by simp


          then show ?thesis using MTArray old' nonLocChanged2_Typed_cond lims 19(7)
            by (simp add: Mutual_NonSub_SpecificNonSub)
        next
          case (MTValue x2)
          then show ?thesis using nonLocChanged2_Typed old 19(7) lims by simp
        qed
      next
        case False
        then have inOld:"Denvalue e $$ ip = Some (type.Memory t2, Stackloc l)"
          using a40 *
          by (metis (lifting) Environment.unfold_congs(5) DenvalueChange fmranE fmranI)
        then have t2Def:"l = (ShowL_nat (Toploc sck')) ∧ t2 = (MTArray x t)" using a40 by simp
        then have acc:"accessStore l k' = Some (KMemptr (ShowL_nat (Toploc mem')))"
          using a40 a20 a1 * unfolding push_def allocate_def updateStore_def accessStore_def by simp
        have "(∀ i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t)"
          using "*" a1 a110 a3 acc cpm2m_TypeCompChangeIndexs cpm2m_def by auto

        moreover have "(∀ v. accessStore ptr_loc m' = Some (MPointer v)
                        ⟶ accessTypeStore ptr_loc m' = Some (MTArray x t))"
          using "*" ‹∀ locations. accessStore locations c' = accessStore locations mem'› a1 a110 acc b27
lNotInC' by auto
        ultimately show ?thesis using t2Def nonLocChanged2_Typed by simp
      qed
    qed

  next
    show "subPrefixStructuralConsistency m'" unfolding subPrefixStructuralConsistency_def
    proof(intros)
      fix locs tp
      assume in1:"accessTypeStore locs m' = Some tp"

      have "∀ locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
      then have nonLocChanged2:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ⟶ accessStore locs
mem' = accessStore locs m'"
        using a3 a1 cpm2m_def[of p l x t mem c'] cpm2mSingleChange[of p l t mem c' x mm'] a2 a110 un-
folding allocate_def by fastforce

      have nonLocChanged_Typed:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)
            ∨ locs = (ShowL_nat (Toploc mem')) ⟶ accessTypeStore locs mem' = accessTypeStore locs
m'"
        using a3 a1 cpm2m_def[of p l x t mem c'] cpm2mSingleChange_typedPref_typed[of p l t mem c' x
mm'] allocateTypeSameAccess a2 a110
        by (metis aloc)

      have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
        using MCon_cpm2m[of p l t mem c' x mm' ] Mconcd 19(2) b2 a1 lNotInC'
```

```
    by fastforce
  then have mcTop:"MCon (MTArray x t) m' l" using a4 a110 19(7) by blast

  have prefPtrs_imps_Pref:"∀x3. TypedMemSubPrefPtrs mm' x t l x3 ⟶ TypedMemSubPref x3 l (MTArray
x t)"
    using selfPoint_imps_TypedMemSubPref[OF selfPoint] a110 by blast

  show "case accessStore locs m' of None ⇒ False | Some (MValue v) ⇒ ∃val. MCon tp m' locs ∧ tp
= MTValue val ∧ accessTypeStore locs m' = Some tp
   | Some (MPointer p) ⇒ ∃len arr. MCon tp m' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore
(hash p (ShowL_nat i)) m' = Some arr)"
  proof(cases "accessTypeStore locs mem' = Some tp")
    case True
    then have accL:"∃v. accessStore locs mem' = Some v"
      using typeSafeDecl_hyps(3) 19(7) unfolding TypeSafe_def SomeValSomeTyp_def by simp
    have notSub:"¬ LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
    proof
      assume c:"LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
      then have "accessStore locs mem' = None"
        using 19(7) nless_le a1 b105 by presburger
      then show False using accL by auto
    qed
    then have sameACC:" accessStore locs m' = accessStore locs mem'" using nonLocChanged2 19(7) by
simp
    then consider (ptr) p where "accessStore locs m' = Some (MPointer p)"
      | (val) val2 where "accessStore locs m' = Some (MValue val2)"
      using typeSafeDecl_hyps(1) 19(7) True unfolding TypeSafe_def subPrefixStructuralConsistency_def
      by (metis memoryvalue.exhaust accL)
    then show ?thesis
    proof(cases)
      case ptr
      then obtain len arr where
        old:"MCon tp mem' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat
i)) mem' = Some arr)"
        using typeSafeDecl_hyps(3) 19(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
      then have lim2:"∀len arr loc. tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem' len arr p loc
                  ⟶ ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) loc"
        using AccessedMemPtrsCantTop[of "mem'" tp p] typeSafeDecl_hyps(3) 19(7) unfolding
TypeSafe_def by blast
      have lim:" ¬ LSubPrefL2 p (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc mem'))
p"
        using AllPtrsNotTop_MCon notSub ptr using typeSafeDecl_hyps(3) 19(7) old unfolding
TypeSafe_def by blast
      have accTLoc:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∨ loc =
          (ShowL_nat (Toploc mem')) ⟶ accessStore loc m' = accessStore loc mem'"
        using 19(7) nonLocChanged2 a1 a110 b27 by presburger
      have "MCon tp m' p" using MCon_memory_transfer[OF _ lim lim2 accTLoc] old by blast
      moreover have "∀i<len. ¬ LSubPrefL2 (hash p (ShowL_nat i)) (ShowL_nat (Toploc mem'))" using
lim
        using MemLSubPrefL2_specific_imps_general by blast
      ultimately have " MCon tp m' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p
(ShowL_nat i)) m' = Some arr)"
        using nonLocChanged2_Typed old 19(7) by simp

      then show ?thesis using ptr by auto
    next
      case val
      then have old:"∃val. MCon tp mem' locs ∧ tp = MTValue val ∧ accessTypeStore locs mem' =
Some tp"
        using typeSafeDecl_hyps(3) 19(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
      then have "∃val. MCon tp m' locs ∧ tp = MTValue val ∧ accessTypeStore locs m' = Some tp"
```

```
          using nonLocChanged2_Typed MCon.simps val notSub 19(7) sameACC by fastforce
        then show ?thesis using val by auto
      qed
    next
      case False
      then have sub:"TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t) ∧ locs ≠ (ShowL_nat
(Toploc mem'))"
        using nonLocChanged_Typed 19(7) in1 by fastforce
      then have "TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)"
        using selfPoint_imps_TypedMemSubPref by simp
      then obtain st where stD:"(CompMemType m' x t st (ShowL_nat (Toploc mem')) locs ∧
         (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash locs
(ShowL_nat i)) m' = Some parent_arr
          | MTValue pval ⇒ accessTypeStore locs m' = Some (MTValue pval)))" using mtp a1 by blast
      then have cmp:"CompMemType m' x t st (ShowL_nat (Toploc mem')) locs " by blast

      have mcLocs:"MCon st m' locs" using cmp mcTop CompTypeRemainsMCon a1 by auto

      show ?thesis
      proof(cases st)
        case (MTArray x11 x12)
        have subs:"t = MTArray x11 x12 ∧ (∃i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat
i)) m' = Some (MPointer locs)) ∨
                    (∃midP subL subA i. CompMemType m' x t (MTArray subL subA) (ShowL_nat (Toploc
mem')) midP
                    ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧
subA = MTArray x11 x12)"
          using CompMemType_imps_Mid[of m' x t x11 x12 "(ShowL_nat (Toploc mem'))" locs] cmp MTArray
by simp

        then have acc:"accessStore locs m' = Some (MPointer locs)"
          using selfPoint mcLocs sub MTArray MCon_imps_TypedMemSubPref_Some mcTop a1
          by (metis MCon.simps(2) a110 option.distinct(1))

        moreover have "∃len arr. MCon tp m' locs ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore
(hash locs (ShowL_nat i)) m' = Some arr)
                          ∧ (∀v. accessStore locs m' = Some (MPointer v) ⟶ accessTypeStore locs m'
= Some (MTArray len arr))"
          using in1
        proof(cases "(∃i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some
(MPointer locs) ∧ t = MTArray x11 x12)")
          case True
          then obtain i where idef:"i<x ∧ accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))
m' = Some (MPointer locs)
                                ∧ t = MTArray x11 x12"
            by auto
          then have "TypedMemSubPref (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) (ShowL_nat (Toploc
mem')) (MTArray x t) " by auto
          then have locsD:"locs = (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))" using selfPoint idef
a1 a110 by blast
          have "∀i<x. accessTypeStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some (MTArray
x11 x12)"
            using idef a1 a110 cpm2m_TypeCompChangeIndexs
            by (metis a3 cpm2m_def)
          then have "tp = MTArray x11 x12" using in1 locsD idef by simp
          moreover have "(∀i<x11. accessTypeStore (hash locs (ShowL_nat i)) m' = Some x12)" using
stD MTArray by simp
          ultimately show ?thesis using mcLocs MTArray in1 by blast
        next
          case False
          then obtain midP i subL subA where mid:"(CompMemType m' x t (MTArray subL subA) (ShowL_nat
(Toploc mem')) midP
                ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
```

```
            using subs by blast
          then have locD:"locs = (hash midP (ShowL_nat i))" using selfPoint a110 a1
            using CompMemType_imps_in_GetAllMemoryLocations_ptr mcTop memSet_selfPoint by blast
          then obtain st2 where st2D:"(CompMemType m' x t st2 (ShowL_nat (Toploc mem')) midP ∧
           (case st2 of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash midP
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore midP m' = Some (MTValue pval)))" using mid mtp a1 a110
            using CompMemTypeSameLocsSameType CompMemType_imps_TypedMemSubPrefPtrs prefPtrs_imps_Pref
by blast
          then have "st2 = (MTArray subL subA)" using mid CompMemTypeSameLocsSameType mcTop a1 by
blast
          then have " ∀i<subL. accessTypeStore (hash midP (ShowL_nat i)) m' = Some subA" using st2D
by simp
          then have "MTArray x11 x12 = tp" using in1 locD mid by simp
          then show ?thesis using mcLocs MTArray stD in1 by fastforce
        qed
        ultimately show ?thesis by auto
      next
        case (MTValue x2)
        then obtain v where vDef:"accessStore locs m' = Some (MValue v)"
          using mcLocs MCon.simps(1)[of x2 m' locs] by (auto split:option.splits memoryvalue.splits)
        moreover have "accessTypeStore locs m' = Some (MTValue x2)" using stD MTValue by auto
        moreover have "tp = MTValue x2" using calculation in1 by simp
        moreover have "MCon tp m' locs" using mcLocs calculation MTValue by blast
        ultimately show ?thesis using MTValue by simp
      qed
    qed
  qed
next
  have old:"SomeValSomeTyp mem'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by blast
  have non:"∀loc. LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶ accessStore loc mem' = None"
    using 19(7) typeSafeDecl_hyps(3) unfolding TypeSafe_def lessThanTopLocs_def
    by auto
  have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
    using MCon_cpm2m[of p l t mem c' x mm' ] Mconcd 19(2) b2 a4 19(2) lNotInC'
    by fastforce
  then have mcTop:"MCon (MTArray x t) m' l" using a4 a110 19(7) by blast
  have somesomeT:"∀destl'.
       TypedMemSubPref destl' (ShowL_nat (Toploc mem')) (MTArray x t) ⟶
       (∃t. accessStore destl' m' = Some t) = (∃tt. accessTypeStore destl' m' = Some tt)"
    using a3 unfolding cpm2m_def
    using cpm2m_TypeCompChange_somesome[of p l t mem c' x mm'] a4 a1 by blast
  show "SomeValSomeTyp m'" unfolding SomeValSomeTyp_def
  proof intros
    fix locs
    show "(∃t. accessStore locs m' = Some t) = (∃tt. accessTypeStore locs m' = Some tt) "
    proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)")
      case True
      then show ?thesis using somesomeT by simp
    next
      case False
      then have "accessStore locs m' = accessStore locs mem'"
        using "19"(7)
        by (metis ‹∀locations. accessStore locations c' = accessStore locations mem'› a1 a110 a3
cpm2mSingleChange_typedPref cpm2m_def)
      moreover have "accessTypeStore locs m' = accessTypeStore locs mem'"
        using False cpm2mSingleChange_typedPref_typed 19(7) allocateTypeSameAccess
        by (metis a1 a110 a3 aloc cpm2m_def)
      ultimately show ?thesis using old
        by (simp add: SomeValSomeTyp_def)
    qed
  qed
next
  assume notCp:"¬cp"
```

```
    then show "Toploc (Memory st) ≤ Toploc m'" using 19(3) by auto
    then show "ncpDenvalueLimit e ev k' (Stack st) (Memory st)" using 19(3) notCp by simp
    show "ncpOMemInDMem (Memory st) m'" using 19 notCp by blast
    show "ncpElementsNoSubPref (Memory st) m'" using 19(3) notCp by blast
    show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 19(3) notCp by blast
  next
    have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t mem c' ] cpm2mTopLocSame[of p l t
mem c' x mm'] b2 by simp
    then show "Toploc mem' ≤ Toploc m' " using aloc unfolding allocate_def by auto
  qed
qed
```

A memory array initialised from a memory pointer without copying: the variable refers to the existing array.

```
lemma typeSafeDecl_20:
  assumes "tp = type.Memory (MTArray x t)"
    and "Some (v, t'') = Some (KMemptr p, tv0)"
    and "¬ cp"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem''', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = type.Memory (MTArray x t) ∧ 0 < x ∧ Denvalue ev'' $$ ip =
None then Some (cdc', mem''',
        astack_dup ip (type.Memory (MTArray x t)) (KMemptr p) (sck'', ev'')) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 20 = assms

  then have a5:"tv0 = (type.Memory (MTArray x t))" using typeSafeDecl_hyps(4) by (metis
option.distinct(1))
  have b2:"x>0" using 20(8) a5 by (metis not_None_eq)
  have NoneIp:"Denvalue ev'' $$ ip = None" using 20 by (simp split:if_splits)
  then have a10:"Some (c, m', k', e) = Some (cdc', mem''', astack_dup ip (type.Memory (MTArray x t))
(KMemptr p) (sck'', ev''))"
    using 20 typeSafeDecl_hyps b2 a5
    by (metis)
  then have a10:"Some (c, m', k', e) = Some (cdc', mem''', astack_dup ip (type.Memory (MTArray x t))
(KMemptr p) (sck', ev''))"
    using 20(7) by simp
  then have a20:"k' = push (KMemptr p) sck'" using 20 NoneIp by force

  have b3:"Address e = Address ev''" using a10 NoneIp by auto
  have a30:"e = (updateEnv ip (type.Memory (MTArray x t)) (Stackloc (ShowL_nat (Toploc sck'))) ev'')"
using a10 NoneIp by simp

  then have a40:
    "Denvalue e = Denvalue(ev'' (| Denvalue := fmupd ip ((type.Memory (MTArray x t)),(Stackloc (ShowL_nat
(Toploc sck')))) (Denvalue ev'') |))" by simp
  then have a50:"(Denvalue e) $$ ip = Some ((type.Memory (MTArray x t)),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp
  have a4:"Address ev'' = Address e"using a30 by simp

  have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
  then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
    using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
  then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
    using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 20(7) by fastforce
  then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip"
    using a40 lessThanTopLocs_def fmranI by auto
  then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''')
                    ∧ l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
```

```
    using a40 lessThanTopLocs_def fmranI by auto

  have c10:"(∀ tp' locs p i.
        (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'') ∧ accessStore locs sck' = Some
(KMemptr p) ∧ i < Toploc
            (Memory st) ∧ LSubPrefL2 p (ShowL_nat i)⟶
        (∃ tp'' loc2 p' .
            (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
            accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧
                tp'' = tp' ∨ (∃ len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr tp'
                p' p))))"
    using typeSafeDecl_hyps(5) 20(3,7) unfolding ncpDenvalueLimit_def by blast
  have c20:"(∃ stloc tp'' p'.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧
          accessStore stloc (Stack st) = Some (KMemptr p') ∧
          (tp'' = (MTArray x t) ∧ p = p' ∨ (∃ len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType
(Memory st) len arr (MTArray x t) p' p)))"
    using typeSafeDecl_hyps(6) a5 20(2,3,7) by fastforce

  have "KMemptr p = v" using 20 by simp
  then have mconp:" MCon (MTArray x t) (Memory st) p" using typeSafeDecl_hyps(2)
extractValueType.simps 20(2,8)
    by (metis type.simps(19) decl.simps(20) option.discI)
  then obtain x' y where ne2:" accessStore (hash p y) (Memory st) = Some x'" using MCon.simps
    by (meson b2 mcon_accessStore)
  have ne3:"((∀ tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore
loc (Memory st) = None) ∧
   (∀ loc y. accessStore loc (Memory st) = Some y ⟶ (∃ tloc<Toploc (Memory st). LSubPrefL2 loc
(ShowL_nat tloc))))"
    using typeSafeDecl_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have ne4:"∃ y tloc. tloc <Toploc (Memory st) ∧ LSubPrefL2 (hash p y) (ShowL_nat tloc)" using ne2
by blast
  then have ne6:"∃ tloc. tloc <Toploc (Memory st) ∧ LSubPrefL2 p (ShowL_nat tloc)" using
MemLSubPrefL2_specific_imps_general by auto

  have ne5:"(∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
              Toploc (Memory st) ≤ Toploc m' ∧ ncpDenvalueLimit ev'' ev sck' (Stack st) (Memory st) ∧
ncpOMemInDMem (Memory
                  st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st) m'"
    using typeSafeDecl_hyps(3,5) 20(3,7) a10 by blast

  have oldLimit:"((∀ tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None) ∧
   (∀ loc y. accessStore loc (Memory st) = Some y ⟶ (∃ tloc<Toploc (Memory st). LSubPrefL2 loc
(ShowL_nat tloc))))"
    using typeSafeDecl_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def by simp

  show ?thesis unfolding TypeSafe_def
  proof(intros)
    fix locs tp assume "¬cp"
    then show "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using 20 typeSafeDecl_hyps(5) by (simp
split:if_splits)
  next
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 20(7) updateEnvUniqueLocs a10
NoneIp by fastforce
  next
    have b2: "compPointers sck' (Denvalue ev'')" using typeSafeDecl_hyps(3) 20(7) unfolding
TypeSafe_def by auto
    moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    ultimately show "compPointers k' (Denvalue e)"
      using compPointersNonStackUpd[of sck' ev'' e ip " type.Memory (MTArray x t)" k' "KMemptr p"]
        a40 a50 typeSafeDecl_hyps(3) 20(7) b2 a20 a4 by simp
  next
```

```
    show "safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using a30 20(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def by
simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe a20 20(7) by metis
  next
    have "addressFormat (Address ev'')" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    then show "addressFormat (Address e)" using 20(7) a4 by (metis)
  next
    have "addressFormat (Sender ev'')" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    then show "addressFormat (Sender e)" using 20(7) a4 by (metis typeSafeDecl_hyps(4) decl_env)
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c" unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev'')) = (Storage st (Address e))" using a30 by simp
      have a110:"m' = mem'" and a115:"cdc' = c" using a10 20 by simp+
      have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
      then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''" using
lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
      then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
          sck'))))" using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 20(7) by
fastforce
      then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc
sck')))) ⟶ ip''' = ip" using a40 fmranI by auto
      then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip" using a40 fmranI by
          auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr p)" using a20 by
(simp add:push_def allocate_def updateStore_def accessStore_def)

          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = KMemptr p " using a170 True by simp
            have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr p)" using a20 by (simp
add:push_def allocate_def updateStore_def accessStore_def)
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have a200:"tDen = (type.Memory (MTArray x t))" using a90 a50 by simp
            then have a210:"t'' = tDen" using a5 20(2) by simp
            have "v = KMemptr p" using 20(2) by simp
            have a220:"MCon (MTArray x t) (Memory st) (extractValueType v)" using a200 a210
typeSafeDecl_hyps(2) by simp
            then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 20(2)
typeSafeDecl_hyps(3) a110 20(7) typeSafeDecl_hyps(5) 20(3) unfolding TypeSafe_def
              by (metis denvalue.simps(5) Option.option.simps(5) stackvalue.simps(19) type.simps(19)
a220 prod.inject extractValueType.simps(3))
          qed
        next
```

```
          case False
          then have "ip'' ≠ ip" using a160 a90 Stackloc by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Stackloc fmranI a90 False
* by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 20(7) by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'" using a20 False by (simp
add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a115 a100 20(7)
              by (cases a; cases tDen; fastforce)
          qed
        qed
      next
        case (Storeloc x2)
        then have a170:"ip'' ≠ ip" using a40 Storeloc a90 by auto
        then have a180: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Storeloc fmranI a90 by
fastforce
        then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
          using Storeloc a100 20(7) by (cases tDen;force)
      qed
    qed
  next
    have "lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    then show "lessThanTopLocs c" using a10 by simp
  next
    have "lessThanTopLocs mem'" using typeSafeDecl_hyps(3) TypeSafe_def 20(7) by simp
    then show "lessThanTopLocs m'" using a10 20 by simp
  next
    show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
  next
    show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t2 l ptr_loc
      assume *:" (type.Memory t2, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"

      show "case t2 of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
      proof(cases "(type.Memory t2, Stackloc l) |∈| fmran(Denvalue ev')")
        case True
        moreover have sameACC:"accessStore l k' = accessStore l sck'"
          using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
          by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
              fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
        then have old:"(case t2 of
                        MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' =
Some arr)
         | MTValue val ⇒ accessTypeStore ptr_loc mem' = Some (MTValue val))"
          using typeSafeDecl_hyps(1) 20(7) * True TypeSafe_def typeSafeDecl_hyps(3)
denvalueTypeCorrectness_def by fastforce
        have mcOld:"MCon t2 mem' ptr_loc" using * sameACC True typeSafeDecl_hyps(1) 20(7)
          unfolding TypeSafe_def typeCompat_def
          by (metis typeSafeDecl_hyps(3) sameMemTSafe)
        have mm:"mem' = m'" using a10 20(7) by blast
        then show ?thesis
```

```
        proof(cases t2)
          case (MTArray x11 x12)
          have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some x12)" using mm old
MTArray by auto

          then show ?thesis using MTArray by simp
        next
          case (MTValue x2)
          then show ?thesis using a10 20 old by simp
        qed
      next
        case False
        then have "Denvalue e $$ ip = Some (type.Memory t2, Stackloc l)"
          using a40 * 20(7)
          by (metis (no_types, lifting) typeSafeDecl_hyps(4) decl_env_not_i fmranE fmranI)
        then have t2Def:"l = (ShowL_nat (Toploc sck')) ∧ t2 = (MTArray x t)" using a40 by simp
        then have acc:"accessStore l k' = Some (KMemptr p)"
          using a40 a20 * unfolding push_def allocate_def updateStore_def accessStore_def by simp

        obtain stloc tp'' pprnt where old:"(
         (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧
         accessStore stloc (Stack st) = Some (KMemptr pprnt) ∧
         (tp'' =(MTArray x t) ∧ p = pprnt ∨
          (∃ len arr. pprnt ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr (MTArray x
t) pprnt p)))"
          using typeSafeDecl_hyps(6) 20
          by (metis a5 extractValueType.simps(3) option.inject prod.inject)
        then obtain len arr where pprntTP: "tp'' = MTArray len arr" using old by blast
        then have old2:"(∀ i<len. accessTypeStore (hash pprnt (ShowL_nat i)) (Memory st) = Some arr)"
          using typeSafeDecl_hyps(1) old unfolding TypeSafe_def denvalueTypeCorrectness_def by
fastforce

        have ncpOld:"ncpOMemInDMem (Memory st) m'" using ne5 by blast
        have someSome:"∀ loc. (∃ t. accessStore loc (Memory st) = Some t) ⟷ (∃ tt. accessTypeStore loc
(Memory st) = Some tt)"
          using typeSafeDecl_hyps(1) unfolding TypeSafe_def SomeValSomeTyp_def by simp
        have ptrEq:"ptr_loc = p" using * acc by simp

        have "(∀ i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t)
              "
        proof(cases "pprnt = p")
          case True
          then have sameT:"tp'' = MTArray x t ∧ len = x ∧ arr = t" using pprntTP old by simp
          then have t'Def: "(∀ i<len. accessTypeStore (hash pprnt (ShowL_nat i)) (Memory st) = Some
arr)"
            using old2 True ptrEq by blast

          show ?thesis
            using sameT ncpOld ne6 old2 ptrEq someSome oldLimit unfolding ncpOMemInDMem_def
            by (metis True)
        next
          case False
          then have comp:"CompMemType (Memory st) len arr (MTArray x t) pprnt ptr_loc" using old
            using pprntTP ptrEq by auto
          then have mids:"arr = MTArray x t ∧ (∃ i<len. accessStore (hash pprnt (ShowL_nat i)) (Memory
st) = Some (MPointer ptr_loc)) ∨
                       (∃ midP subL subA i.
                     CompMemType (Memory st) len arr (MTArray subL subA) pprnt midP
                     ∧ accessStore (hash midP (ShowL_nat i)) (Memory st) = Some (MPointer ptr_loc) ∧ i <
subL ∧ subA = MTArray x t)"
            using CompMemType_imps_Mid[OF comp] by simp
          then have mcPtrLoc:"MCon (MTArray x t) (Memory st) ptr_loc "
            using mconp ptrEq by blast
          have occ:"(∀ locs tp.
```

```
        accessTypeStore locs (Memory st) = Some tp ⟶
        (case accessStore locs (Memory st) of None ⇒ False
         | Some (MValue v) ⇒ ∃val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs
(Memory st) = Some tp
         | Some (MPointer p) ⇒ ∃len arr. MCon tp (Memory st) p ∧ tp = MTArray len arr
            ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some arr)
            ))"
            using typeSafeDecl_hyps(1) unfolding TypeSafe_def subPrefixStructuralConsistency_def by
auto

          have accO:"(∀i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some t)"
          proof (cases "arr = MTArray x t ∧ (∃i<len. accessStore (hash pprnt (ShowL_nat i)) (Memory st)
= Some (MPointer ptr_loc))")
            case True
            then obtain ii where "ii<len ∧ accessStore (hash pprnt (ShowL_nat ii)) (Memory st) = Some
(MPointer ptr_loc)" by blast
            then have "∃len arr'. MCon arr (Memory st) ptr_loc ∧ arr = MTArray len arr'
                        ∧ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr')
                        "
              using old2 occ by fastforce
            then show ?thesis using True by blast
          next
            case False
            then obtain midP subL subA i where iidef:"(CompMemType (Memory st) len arr (MTArray subL
subA) pprnt midP
                    ∧ accessStore (hash midP (ShowL_nat i)) (Memory st) = Some (MPointer ptr_loc) ∧ i <
subL ∧ subA = MTArray x t)"
              using mids by blast
            then have "accessTypeStore (hash midP (ShowL_nat i)) (Memory st) = Some (MTArray x t)"
              by (metis (no_types, lifting) typeSafeDecl_hyps(1) CompMemTypeSubIndexes old old2 pprntTP
TypeSafe_def sameMemTSafe)
            then show ?thesis using occ iidef by fastforce
          qed


          then have "(∀i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t)"
            using ncpOld ne6 ptrEq someSome oldLimit unfolding ncpOMemInDMem_def by metis
          then show ?thesis by simp
        qed

        then show ?thesis using t2Def by simp
      qed
    qed
  next
    have sameMem:"m' = mem'" using 20(7,8) by (auto split:if_splits)
    then show "subPrefixStructuralConsistency m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def
by blast
  next
    show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a10 20(7) by
blast
  next

    assume notCp:"¬cp"
    then show "Toploc (Memory st) ≤ Toploc m'" using 20(7) a10 typeSafeDecl_hyps by auto

    show "ncpOMemInDMem (Memory st) m'" using 20(7) a10 typeSafeDecl_hyps notCp by blast
    show "ncpElementsNoSubPref (Memory st) m'" using 20(7) a10 typeSafeDecl_hyps notCp by blast
    show ncp:"ncpDenvalueLimit e ev k' (Stack st) (Memory st)" unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p''' i''
      assume asm:"(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' = Some
(KMemptr p''')
                ∧ i'' < Toploc (Memory st) ∧ LSubPrefL2 p''' (ShowL_nat i'')"
      then obtain i where idef:"Denvalue e $$ i = Some (type.Memory tp', Stackloc locs)" by blast
```

```
      have "ncpDenvalueLimit ev'' ev sck' (Stack st) (Memory st)" using 20 typeSafeDecl_hyps(5) by
blast
      then have old:"∀tp' locs p i.
     (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'') ∧ accessStore locs sck' = Some (KMemptr
p) ∧ i < Toploc
         (Memory st) ∧ LSubPrefL2 p (ShowL_nat i) ⟶
     (∃tp'' loc2 p'.
         (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
         accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧ tp'' = tp' ∨ (∃len arr. p' ≠ p ∧
tp''
             = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p)))"
        unfolding ncpDenvalueLimit_def by blast
      show "∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p''' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'''
              ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p'''))"
      proof(cases "locs = (ShowL_nat (Toploc sck'))")
        case True
        then have b45:"tp' = (MTArray x t)" using a150 a50 True
          using idef by fastforce
        have b50:"p''' = p" using a40 a20 push_def
          by (metis stackvalue.inject(3) True accessStore_def accessStore_updateStore allocateMapping
asm old.prod.exhaust option.inject snd_eqD)

        have c20:"(∃stloc tp'' p'.
           (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue ev) ∧
           accessStore stloc (Stack st) = Some (KMemptr p') ∧
           (tp'' = (MTArray x t) ∧ p = p' ∨ (∃len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType
(Memory st) len arr (MTArray x t) p' p)))"
          using typeSafeDecl_hyps a5 20(2,3,7) by fastforce
        then show ?thesis using old a40 a20 asm b45 b50 by blast
      next
        case False
        then have b5:"locs ≠ (ShowL_nat (Toploc sck'))" using a20 a40 asm
          by (metis)
        then have "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'')" using a40 a160 fmranI
asm by fastforce
        moreover have "accessStore locs sck' = accessStore locs k'" using b5 a20 unfolding push_def
accessStore_def updateStore_def allocate_def by auto
        ultimately show ?thesis using old asm by metis
      qed
    qed
  next
    assume notCP:"¬cp"
    then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 20 a10 by blast
  next
    show "Toploc mem' ≤ Toploc m'" using a10 20 by blast
  qed
qed
```

A memory array initialised from a calldata pointer: the array is copied into fresh memory.

```
lemma typeSafeDecl_21:
  assumes "tp = type.Memory (MTArray x t)"
    and "Some (v, t'') = Some (KCDptr p, tv0)"
    and "cp = cp0"
    and "cd = cdc"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem''', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = Calldata (MTArray x t) ∧ 0 < x ∧ Denvalue
        ev'' $$ ip = None then cpm2m p (ShowL_nat (Toploc mem''')) x t cdc (snd (allocate
        mem''')) ≫= (λm'. Some (cdc', m', astack_dup ip (type.Memory (MTArray x t)) (KMemptr (ShowL_nat
```

```
(Toploc mem'''))) (sck'', ev''))) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 21 = assms
  then obtain l where a1:"l = ShowL_nat(Toploc mem')" by simp
  have vk:"tv0 = Calldata (MTArray x t)" using decl.simps(9) 21(8) by (simp split:if_split_asm)
  then have locationscd':"(∀ tloc loc. Toploc mem' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc mem' = None)"
    using typeSafeDecl_hyps(3) 21 unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have b1:"accessStore l mem' = None" using a1 using Read_Show_nat'_id using LSubPrefL2_def by
auto
  have b2:"x>0" using 21(8) vk a1 by (metis not_None_eq)
  then have Mconcd:"MCon (MTArray x t) cdc (extractValueType v)" using typeSafeDecl_hyps(2) 21(2)
21(4) vk 21(5) by simp
  have NoneIp:"Denvalue ev'' $$ ip = None" using 21 by (simp split:if_split_asm)
  obtain c' where a2:"∃dud. (dud, c') = allocate mem'" by (simp add: allocate_def)
  then have aloc:"snd (allocate mem') = c'" by (simp add: allocate_def)
  then have lNotInC':"accessStore l c' = None" using b1 a1 unfolding allocate_def accessStore_def by
auto
  obtain mm' where a3:"cpm2m p l x t cdc c' = Some mm'" using 21 a1 a2
    using b2 vk aloc NoneIp by fastforce
  then have a4:"Some (c, m', k', e) = Some (cdc', mm', astack ip (type.Memory (MTArray x t)) (KMemptr
l) (sck', ev''))"
    using a1 a2 a3 21 decl.simps(9) vk b2 aloc NoneIp
    by simp
  then have a20:"k' = push (KMemptr l) sck'" by force
  have a30:"e = (updateEnv ip (type.Memory (MTArray x t)) (Stackloc (ShowL_nat (Toploc sck'))) ev'')"
using a4 by simp
  then have a40:"Denvalue e = Denvalue(ev'' (| Denvalue := fmupd ip (type.Memory (MTArray x t),(Stackloc
(ShowL_nat (Toploc sck')))) (Denvalue ev'') |))" by simp
  then have a50:"(Denvalue e) $$ ip = Some (type.Memory (MTArray x t),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp

  then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
    using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] by fastforce
  have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2 lNotInC' a1
by auto
  have "∀ locations. accessStore locations c' = accessStore locations mem'" using a2 accessAllocate[of
c' mem'] by auto
  then have b110:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs mm'"
    using b100 a2 unfolding allocate_def by simp
  then have subMapping:"Mapping mem' ⊆_f Mapping mm'" using b105 unfolding fmsubset_alt_def
    by (metis (mono_tags, lifting) accessStore_def fmpredI option.distinct(1))

  have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
  then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v''''"
    using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
  then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
    using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 21(7) by fastforce
  then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip"
    using a40 lessThanTopLocs_def fmranI by auto
  then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''')
                  ∧ l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
    using a40 lessThanTopLocs_def fmranI by auto
  have mcTop:"MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def a4
    using MCon_cpm2m[of p l t cdc c' x mm' ] Mconcd 21(2) b2 lNotInC' by fastforce
  have a110:"mm' = m'" using a4 by simp

  then have selfPoint:"∀ la l'. TypedMemSubPref la l (MTArray x t) ∧ accessStore la m' = Some (MPointer
l') ⟶ l' = la"
    using cpm2mSelfPointers[of p l t cdc c' x mm'] a3 a1 cpm2m_def[of p l x t cdc c'] by metis
```

```
  have "∀locations. accessStore locations c' = accessStore locations mem'" using a2 accessAllocate[of
c' mem'] by auto
  then have b27:"∀locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStore locs mem' = accessStore locs
mm'"
    using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] a2 unfolding
allocate_def by fastforce
  then have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                          ⟶ accessTypeStore locs c' = accessTypeStore locs m'"
    using cpm2mSingleChange_Typed[of ] allocateTypeSameAccess a3 a1 a110 unfolding cpm2m_def by
presburger
  moreover have "∀locs. accessTypeStore locs c' = accessTypeStore locs mem'" using aloc
allocateTypeSameAccess by metis
  ultimately have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                        ⟶ accessTypeStore locs mem' = accessTypeStore locs m'" by
simp
  have "∀destl'.
       TypedMemSubPref destl' l (MTArray x t) ⟶
       (∃st. CompMemType mm' x t st l destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) mm' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' mm' = Some (MTValue pval)))"
    using a1 a3 unfolding cpm2m_def using cpm2m_TypeCompChange[of p l t cdc c' x mm'] by blast
  then have mtp:"∀destl'. TypedMemSubPref destl' l (MTArray x t) ⟶ (∃st. CompMemType m' x t st l
destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval)))" using a110 by blast

  show ?thesis unfolding TypeSafe_def
  proof(intros)
    have a110:"m' = mm'" using a4 by simp
    then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
      using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] by fastforce
    have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2 lNotInC'
a1 by auto
    have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
    then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def by simp

    fix locs tp assume ncp:"¬cp"
    then have a120:"∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs" using
typeSafeDecl_hyps(5) 21(7) by simp
    moreover have "∀locs tp. MCon tp mem' locs ⟶ MCon tp mm' locs" using mconCopySingle[of l mem'
mm' ] using b110 b105 by simp
    moreover have "Toploc c' > Toploc mem'" using a2 unfolding allocate_def by simp
    moreover have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ]
cpm2mTopLocSame[of p l t cdc c' x mm'] b2 by simp
    ultimately show "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using 21 typeSafeDecl_hyps(5) 21(7)
a110 a120 ncp by blast
  next
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 21(7) updateEnvUniqueLocs a4 by
blast
  next
    have b2: "compPointers sck' (Denvalue ev'')" using typeSafeDecl_hyps(3) 21(7) unfolding
TypeSafe_def by auto
    have "Address ev'' = Address e"using a30 by simp
    moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
    ultimately show "compPointers k' (Denvalue e)"
      using compPointersNonStackUpd[of sck' ev'' e ip " type.Memory (MTArray x t)" k' "KMemptr l" ]
        a40 a50 typeSafeDecl_hyps(3) 21(7) b2 a20 by simp
  next
```

```
    show "safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using a30 21(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def by
simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe a20 21(7) by metis
  next
    have "addressFormat (Address ev'') " using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
    then show "addressFormat(Address e) " using 21(7) a4 by auto
  next
    have "addressFormat(Sender ev'')" using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
    then show "addressFormat (Sender e)" using 21(7) a4 by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c" unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev'')) = (Storage st (Address e))" using a30 by simp
      have a110:"m' = mm'" using a4 by simp
      have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 21(7) by simp
      then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
        using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
      then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
        using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 21(7) by fastforce
      then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc
sck')))) ⟶ ip''' = ip"
        using a40 lessThanTopLocs_def fmranI by auto
      then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''')
                        ∧ l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using a40 lessThanTopLocs_def fmranI by auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by
(simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = KMemptr l " using a170 True by simp
            have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by (simp
add:push_def allocate_def updateStore_def accessStore_def)
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have a200:"tDen = type.Memory (MTArray x t)" using a90 a50 by simp
            have a220:"v = KCDptr p" using 21(2) by simp
            have "c = cdc'" using a4 by simp
            then have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
              using MCon_cpm2m[of p l t cdc c' x mm' ] Mconcd 21(2) b2
              by (metis a220 lNotInC' extractValueType.simps(2))
            then have "MCon (MTArray x t) m' l" using a4 a110 21(7) by blast
            then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 21(2) by simp
          qed
        next
          case False
```

```
          then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
            using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] by
fastforce

          have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2
lNotInC' a1 by auto
          have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
          then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs
mm'"
            using b100 a2 unfolding allocate_def by simp

          then have "ip'' ≠ ip" using a160 a90 Stackloc False by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Stackloc fmranI a90 False
* by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 21 by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'"
            using a20 False by (simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis
            proof(cases a)
              case (KValue x1)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
                using a190 some Stackloc a170 * a180 a110 a100 21(7) by (cases tDen; fastforce)
            next
              case (KCDptr x2)
              then have b10:"(case tDen of Calldata struct ⇒ MCon struct cdc' x2 | _ ⇒ False)"
                using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 21(7) unfolding TypeSafe_def
typeCompat_def by force
              then obtain struct where tden: "tDen = Calldata struct" by (auto split:type.splits)
              then have "MCon struct cdc' x2" using b10 by simp
              then show ?thesis using Stackloc some KCDptr tden a110 a4 by simp
            next
              case (KMemptr x3)
              then have b10:"(case tDen of type.Memory struct ⇒ MCon struct mem' x3 | _ ⇒ False)"
                using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 21(7) unfolding TypeSafe_def
typeCompat_def by force
              then obtain struct where tden: "tDen = type.Memory struct" by (auto split:type.splits)
              then have "MCon struct mem' x3" using b10 by simp
              then have "MCon struct mm' x3" using mconCopySingle[of l mem' mm' struct] b110 b105 by
auto
              then show ?thesis using Stackloc some KMemptr tden a110 a4 by simp
            next
              case (KStoptr x4)
              then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
                using a190 some Stackloc a170 * a180 a110 a100 21(7) by (cases tDen; fastforce)
            qed
          qed
        qed
      next
        case (Storeloc x2)
        then have "ip'' ≠ ip" using a160 a90 by simp
        then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Storeloc fmranI a90 * by
fastforce
        then have a180:"(case tDen of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev'')) | _
⇒ False)"
          using a170 Storeloc using typeSafeDecl_hyps(3) 21(7)
          unfolding TypeSafe_def typeCompat_def by force
```

```
        then obtain typ' where tdent:"tDen = type.Storage typ'" by (auto split: type.splits)
        then have "SCon typ' x2 (Storage st (Address ev''))" using a180 by simp
        then have "SCon typ' x2 (Storage st (Address e))" using a100 by simp
        then show ?thesis using a170 Storeloc using typeSafeDecl_hyps(3) 21(7) tdent
          by simp
      qed
    qed
  next
    have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
cdc c' x mm'] b2 by simp
    show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t2 l ptr_loc
      assume *:" (type.Memory t2, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"

      show "case t2 of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
      proof(cases "(type.Memory t2, Stackloc l) |∈| fmran(Denvalue ev')")
        case True
        moreover have sameACC:"accessStore l k' = accessStore l sck'"
          using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
          by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
              fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
        then have old:"(case t2 of
                        MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' =
Some arr)
                        | MTValue val ⇒ accessTypeStore ptr_loc mem' = Some (MTValue val))"
          using typeSafeDecl_hyps(1) 21(7) * True typeSafeDecl_hyps(3) unfolding TypeSafe_def
denvalueTypeCorrectness_def by simp
        have mcOld:"MCon t2 mem' ptr_loc" using * sameACC True typeSafeDecl_hyps(1) 21(7)
          unfolding TypeSafe_def typeCompat_def
          by (metis typeSafeDecl_hyps(3) sameMemTSafe)

        have lims:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) ptr_loc"
          using typeSafeAllPtrsNotTop[OF typeSafeDecl_hyps(3) True] * 21(7) sameACC b99 aloc unfolding
allocate_def
          using LSubPrefL2_def MemLSubPrefTransitive by force
        then show ?thesis
        proof(cases t2)
          case (MTArray x11 x12)
          have nonLocChanged2_Typed_cond:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶
accessTypeStore loc m' = accessTypeStore loc mem'"
            using nonLocChanged2_Typed by simp
          have nonLochanged:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶ accessStore loc m' =
accessStore loc mem'"
            using b27 a1 a110 by simp
          have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' = Some x12)"
            using old MTArray by simp
          show ?thesis using MTArray old' nonLocChanged2_Typed_cond lims 21(7)
            using MemLSubPrefL2_specific_imps_general mtypes.simps(5) old nonLochanged
            by (metis (no_types, lifting) ext)
        next
          case (MTValue x2)
          then show ?thesis using nonLocChanged2_Typed old 21(7) lims by simp
        qed
      next
        case False
        then have "Denvalue e $$ ip = Some (type.Memory t2, Stackloc l)"
          using a40 * a4
          using typeSafeDecl_hyps(4) decl_env_not_i fmlookup_ran_iff by fastforce
```

```
        then have t2Def:"l = (ShowL_nat (Toploc sck')) ∧ t2 = (MTArray x t)" using a40 by simp
        then have acc:"accessStore l k' = Some (KMemptr (ShowL_nat (Toploc mem')))"
          using a40 a20 a1 * unfolding push_def allocate_def updateStore_def accessStore_def by simp
        have "(∀i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t)"
          using "*" a1 a110 a3 acc cpm2m_TypeCompChangeIndexs cpm2m_def by auto
        then show ?thesis using t2Def
          using "*" a1 a110 acc b1 b27 by fastforce
      qed
    qed
  next
    show "subPrefixStructuralConsistency m'" unfolding subPrefixStructuralConsistency_def
    proof(intros)
      fix locs tp
      assume in1:"accessTypeStore locs m' = Some tp"

      have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
      then have nonLocChanged2:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ⟶ accessStore locs
mem' = accessStore locs m'"
        using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] a2 a110 un-
folding allocate_def by fastforce

      have nonLocChanged_Typed:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)
            ∨ locs = (ShowL_nat (Toploc mem')) ⟶ accessTypeStore locs mem' = accessTypeStore locs
m'"
        using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange_typedPref_typed[of p l t cdc c' x
mm'] allocateTypeSameAccess a2 a110
        by (metis aloc)

      have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
        using MCon_cpm2m[of p l t cdc c' x mm' ] Mconcd 21(2) b2 a1 lNotInC'
        by fastforce
      then have mcTop:"MCon (MTArray x t) m' l" using a4 a110 21(7) by blast

      have prefPtrs_imps_Pref:"∀x3. TypedMemSubPrefPtrs mm' x t l x3 ⟶ TypedMemSubPref x3 l (MTArray
x t)"
        using selfPoint_imps_TypedMemSubPref[OF selfPoint] a110 by blast

      show "case accessStore locs m' of None ⇒ False | Some (MValue v) ⇒ ∃val. MCon tp m' locs ∧ tp
= MTValue val ∧ accessTypeStore locs m' = Some tp
       | Some (MPointer p) ⇒ ∃len arr. MCon tp m' p ∧ tp = MTArray len arr
∧ (∀i<len. accessTypeStore (hash p (ShowL_nat i)) m' = Some arr) "
      proof(cases "accessTypeStore locs mem' = Some tp")
        case True
        then have accL:"∃v. accessStore locs mem' = Some v"
          using typeSafeDecl_hyps(3) 21(7) unfolding TypeSafe_def SomeValSomeTyp_def by simp
        have notSub:"¬ LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
        proof
          assume c:"LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
          then have "accessStore locs mem' = None"
            using 21(7) nless_le a1 b105 by presburger
          then show False using accL by auto
        qed
        then have sameACC:" accessStore locs m' = accessStore locs mem'" using nonLocChanged2 21(7) by
simp
        then consider (ptr) p where "accessStore locs m' = Some (MPointer p)"
          | (val) val2 where "accessStore locs m' = Some (MValue val2)"
          using typeSafeDecl_hyps(1) 21(7) True unfolding TypeSafe_def subPrefixStructuralConsistency_def
          by (metis memoryvalue.exhaust accL)
        then show ?thesis
        proof(cases)
          case ptr
          then obtain len arr where
            old:"MCon tp mem' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat
```

```
i)) mem' = Some arr) "
            using typeSafeDecl_hyps(3) 21(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
          then have lim2:"∀ len arr loc. tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem' len arr p loc
                      ⟶ ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) loc"
            using AccessedMemPtrsCantTop[of "mem'" tp p] typeSafeDecl_hyps(3) 21(7) unfolding
TypeSafe_def by blast
          have lim:" ¬ LSubPrefL2 p (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc mem'))
p"
            using AllPtrsNotTop_MCon notSub ptr using typeSafeDecl_hyps(3) 21(7) old unfolding
TypeSafe_def by blast
          have accTLoc:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∨ loc =
              (ShowL_nat (Toploc mem')) ⟶ accessStore loc m' = accessStore loc mem'"
            using 21(7) nonLocChanged2 a1 a110 b27 by presburger
          have "MCon tp m' p" using MCon_memory_transfer[OF _ lim lim2 accTLoc] old by blast
          moreover have "∀ i<len. ¬ LSubPrefL2 (hash p (ShowL_nat i)) (ShowL_nat (Toploc mem'))" using
lim
            using MemLSubPrefL2_specific_imps_general by blast
          ultimately have "MCon tp m' p ∧ tp = MTArray len arr ∧ (∀ i<len. accessTypeStore (hash p
(ShowL_nat i)) m' = Some arr)"
            using nonLocChanged2_Typed old 21(7) by simp

          then show ?thesis using ptr by auto
        next
          case val
          then have old:"∃ val. MCon tp mem' locs ∧ tp = MTValue val ∧ accessTypeStore locs mem' =
Some tp"
            using typeSafeDecl_hyps(3) 21(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
          then have "∃ val. MCon tp m' locs ∧ tp = MTValue val ∧ accessTypeStore locs m' = Some tp"
            using nonLocChanged2_Typed MCon.simps val notSub 21(7) sameACC by fastforce
          then show ?thesis using val by auto
        qed
      next
        case False
        then have sub:"TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t) ∧ locs ≠ (ShowL_nat
(Toploc mem'))"
          using nonLocChanged_Typed 21(7) in1 by fastforce
        then have "TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)"
          using selfPoint_imps_TypedMemSubPref by simp
        then obtain st where stD:"(CompMemType m' x t st (ShowL_nat (Toploc mem')) locs ∧
           (case st of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash locs
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore locs m' = Some (MTValue pval)))" using mtp a1 by blast
        then have cmp:"CompMemType m' x t st (ShowL_nat (Toploc mem')) locs " by blast

        have mcLocs:"MCon st m' locs" using cmp mcTop CompTypeRemainsMCon a1 by auto

        show ?thesis
        proof(cases st)
          case (MTArray x11 x12)
          have subs:"t = MTArray x11 x12 ∧ (∃ i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat
i)) m' = Some (MPointer locs)) ∨
                     (∃ midP subL subA i. CompMemType m' x t (MTArray subL subA) (ShowL_nat (Toploc
mem')) midP
                     ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧
subA = MTArray x11 x12)"
            using CompMemType_imps_Mid[of m' x t x11 x12 "(ShowL_nat (Toploc mem'))" locs] cmp MTArray
by simp

          then have acc:"accessStore locs m' = Some (MPointer locs)"
            using selfPoint mcLocs sub MTArray MCon_imps_TypedMemSubPref_Some mcTop a1
            by (metis MCon.simps(2) option.distinct(1))
```

```
        moreover have "∃len arr. MCon tp m' locs ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore
(hash locs (ShowL_nat i)) m' = Some arr)
                          ∧ (∀v. accessStore locs m' = Some (MPointer v) ⟶ accessTypeStore locs m'
= Some (MTArray len arr))"
          using in1
        proof(cases "(∃i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some
(MPointer locs) ∧ t = MTArray x11 x12)")
          case True
          then obtain i where idef:"i<x ∧ accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))
m' = Some (MPointer locs)
                                ∧ t = MTArray x11 x12"
            by auto
          then have "TypedMemSubPref (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) (ShowL_nat (Toploc
mem')) (MTArray x t) " by auto
          then have locsD:"locs = (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))" using selfPoint idef
a1 a110 by blast
          have "∀i<x. accessTypeStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some (MTArray
x11 x12)"
            using idef a1 a110 cpm2m_TypeCompChangeIndexs
            by (metis a3 cpm2m_def)
          then have "tp = MTArray x11 x12" using in1 locsD idef by simp
          moreover have "(∀i<x11. accessTypeStore (hash locs (ShowL_nat i)) m' = Some x12)" using
stD MTArray by simp
          ultimately show ?thesis using mcLocs MTArray in1 by blast
        next
          case False
          then obtain midP i subL subA where mid:"(CompMemType m' x t (MTArray subL subA) (ShowL_nat
(Toploc mem')) midP
                ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
            using subs by blast
          then have locD:"locs = (hash midP (ShowL_nat i))" using selfPoint a110 a1
            using CompMemType_imps_in_GetAllMemoryLocations_ptr mcTop memSet_selfPoint by blast
          then obtain st2 where st2D:"(CompMemType m' x t st2 (ShowL_nat (Toploc mem')) midP ∧
           (case st2 of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash midP
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore midP m' = Some (MTValue pval)))" using mid mtp a1 a110
            using CompMemTypeSameLocsSameType CompMemType_imps_TypedMemSubPrefPtrs prefPtrs_imps_Pref
by blast
          then have "st2 = (MTArray subL subA)" using mid CompMemTypeSameLocsSameType mcTop a1 by
blast
          then have " ∀i<subL. accessTypeStore (hash midP (ShowL_nat i)) m' = Some subA" using st2D
by simp
          then have "MTArray x11 x12 = tp" using in1 locD mid by simp
          then show ?thesis using mcLocs MTArray stD in1 by fastforce
        qed
        ultimately show ?thesis by auto
      next
        case (MTValue x2)
        then obtain v where vDef:"accessStore locs m' = Some (MValue v)"
          using mcLocs MCon.simps(1)[of x2 m' locs] by (auto split:option.splits memoryvalue.splits)
        moreover have "accessTypeStore locs m' = Some (MTValue x2)" using stD MTValue by auto
        moreover have "tp = MTValue x2" using calculation in1 by simp
        moreover have "MCon tp m' locs" using mcLocs calculation MTValue by blast
        ultimately show ?thesis using MTValue by simp
      qed
    qed
  qed
next
  show "lessThanTopLocs c" using typeSafeDecl_hyps(3) 21(7) a4 unfolding TypeSafe_def by auto
next
  have b100:"lessThanTopLocs mem'" using typeSafeDecl_hyps(3) 21(7) unfolding TypeSafe_def by blast
  have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
```

```
cdc c' x mm'] b2 by simp
    have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs m'"
      using cpm2mSingleChange[of p l t cdc c' x mm'] a4 a3 cpm2m_def[of p l x t cdc c'] by fastforce
    moreover have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
    ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs m' = accessStore locs mem'"
by simp
    show "lessThanTopLocs m'" using cdMemLocsToploc[of mem' l c' m'] b100 a1 a2 b99 using b10 by
presburger
  next
    show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
  next
    assume notCP:"¬cp"
    show "ncpDenvalueLimit e ev k' (Stack st) (Memory st)" unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p''' i''
      assume asm:" (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' =
Some (KMemptr p''') ∧ i'' <
          Toploc (Memory st) ∧ LSubPrefL2 p''' (ShowL_nat i'')"
      then obtain i where idef:"Denvalue e $$ i = Some (type.Memory tp', Stackloc locs)" by blast

      have "ncpDenvalueLimit ev'' ev sck' (Stack st) (Memory st)" using 21 notCP typeSafeDecl_hyps(5)
by blast
      then have old:"∀tp' locs p i.
     (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'') ∧ accessStore locs sck' = Some (KMemptr
p) ∧ i < Toploc
         (Memory st) ∧ LSubPrefL2 p (ShowL_nat i) ⟶
     (∃tp'' loc2 p'.
         (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
         accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧ tp'' = tp' ∨ (∃len arr. p' ≠ p ∧
tp''
             = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p)))"
        unfolding ncpDenvalueLimit_def by blast
      show "∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p''' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'''
              ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p'''))"
      proof(cases "locs = (ShowL_nat (Toploc sck'))")
        case True
        then have b45:"tp' = (MTArray x t)" using a150 a50 True
          using idef by fastforce
        have b50:"p''' = l" using a40 a20 push_def
          by (metis stackvalue.inject(3) True accessStore_def accessStore_updateStore allocateMapping
asm old.prod.exhaust option.inject snd_eqD)
        then have " p''' = (ShowL_nat (Toploc mem'))" using a1 by blast
        moreover have "Toploc mem' ≥ Toploc (Memory st)" using 21 notCP typeSafeDecl_hyps(5) by blast
        ultimately show ?thesis using asm
          by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general hash_inequality
hash_suffixes_associative hashesIntSame le_antisym nat_less_le)
      next
        case False
        then have b5:"locs ≠ (ShowL_nat (Toploc sck'))" using a20 a40 asm
          by (metis)
        then have "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'')" using a40 a160 fmranI
asm by fastforce
        moreover have "accessStore locs sck' = accessStore locs k'" using b5 a20 unfolding push_def
accessStore_def updateStore_def allocate_def by auto
        ultimately show ?thesis using old asm by metis
      qed
    qed
  next
    assume notCP:"¬cp"
    show "ncpOMemInDMem (Memory st) m'" unfolding ncpOMemInDMem_def
```

```
    proof intros
      fix i loc
      assume c10:" i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)"
      then have old:" accessStore loc (Memory st) = accessStore loc mem' ∧ accessTypeStore loc (Memory
st) = accessTypeStore loc mem'"
        using typeSafeDecl_hyps(5) 21(7) notCP c10 ncpOMemInDMem_def by simp
      have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs m'"
        using cpm2mSingleChange[of p l t cdc c' x mm'] a4 a3 cpm2m_def[of p l x t cdc c'] by fastforce
      moreover have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
      ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs m' = accessStore locs mem'"
by simp
      have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l
t cdc c' x mm'] b2 by simp
      have "Toploc (Memory st) ≤ Toploc mem'" using typeSafeDecl_hyps(5) c10 21(7) notCP by simp
      then have notI:"(ShowL_nat i) ≠ l" using a1 typeSafeDecl_hyps(5) c10 b99
        by (metis hashesIntSame leD)
      then show "accessStore loc (Memory st) = accessStore loc m'"
        using c10 a1 b99 a2 cdMemLocsToploc[of mem' l c' m']
        using "21"(3)
        by (metis LSubPrefL2_def old b10 hash_suffixes_associative hashesInts)

      have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessTypeStore locs c' = accessTypeStore locs m'"
        using cpm2mSingleChange_Typed[of p l t cdc c' x mm'] a4 a3 cpm2m_def[of p l x t cdc c'] by
fastforce
      moreover have "∀locations. accessTypeStore locations c' = accessTypeStore locations mem'"
        using a2 allocateTypeSameAccess
        by (simp add: <∀locs. accessTypeStore locs c' = accessTypeStore locs mem'>)
      ultimately have "∀locs. ¬ LSubPrefL2 locs l ⟶ accessTypeStore locs m' = accessTypeStore locs
mem'" by simp
      then show " accessTypeStore loc (Memory st) = accessTypeStore loc m'"
        using c10 a1 b99 a2 cdMemLocsToploc[of mem' l c' m']
        using LSubPrefL2_def old hash_suffixes_associative hashesInts 21(3) notI by metis
    qed
  next
    assume notCP:"¬cp"
    have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
cdc c' x mm'] b2 by simp

    have a110:"m' = mm'" using a4 by simp
    then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
      using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] by fastforce
    have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2 lNotInC'
a1 by auto
    have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
    then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def by simp

    fix locs tp assume ncp:"¬cp"
    then have a120:"∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs" using
typeSafeDecl_hyps(5) 21(7) by simp
    moreover have "∀locs tp. MCon tp mem' locs ⟶ MCon tp mm' locs" using mconCopySingle[of l mem'
mm' ] using b110 b105 by simp
    moreover have b98:"Toploc c' > Toploc mem'" using a2 unfolding allocate_def by simp
    moreover have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ]
cpm2mTopLocSame[of p l t cdc c' x mm'] b2 by simp
    ultimately have "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using 21 typeSafeDecl_hyps(5) 21(7)
a110 a120 ncp by blast

    have selfPoint:"∀la l'. TypedMemSubPref la l (MTArray x t) ∧ accessStore la mm' = Some (MPointer
l') ⟶ l' = la"
      using cpm2mSelfPointers[of p l t cdc c' x mm'] a3 a1 cpm2m_def[of p l x t cdc c'] by argo
```

```
    have sameaccess:"∀locs. locs ≠ l ∧ ¬ TypedMemSubPref locs l (MTArray x t) ⟶ accessStore locs c'
= accessStore locs mm'"
      using cpm2mSingleChange_typedPref[of p l t cdc c' x mm'] a3 a1 cpm2m_def[of p l x t cdc c'] by
fastforce
    then have sameaccess:"∀locs. locs ≠ l ∧ ¬ TypedMemSubPref locs l (MTArray x t) ⟶ accessStore
locs mem' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def a2 accessAllocate[of c' mem']
      by (simp add: <∀locations. accessStore locations c' = accessStore locations mem'>)

    have limitsOld:" ((∀tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None) ∧
     (∀loc y. accessStore loc (Memory st) = Some y ⟶ (∃tloc<Toploc (Memory st). LSubPrefL2 loc
(ShowL_nat tloc))))"
      using typeSafeDecl_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def using 21(7) by auto

    have limits:"((∀tloc loc. Toploc mem' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
mem' = None) ∧
                    (∀loc y. accessStore loc mem' = Some y ⟶ (∃tloc<Toploc mem'. LSubPrefL2 loc
(ShowL_nat tloc))))"
      using typeSafeDecl_hyps(3) 21(7) unfolding TypeSafe_def lessThanTopLocs_def by simp

    have old:"ncpElementsNoSubPref (Memory st) mem' ∧ ncpOMemInDMem (Memory st) mem' ∧ ncpNewSelfPoint
(Memory st) mem'" using 21 notCP typeSafeDecl_hyps(5) by blast
    then have sameLocs:"(∀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
        accessStore loc (Memory st) = accessStore loc mem')" unfolding ncpOMemInDMem_def by blast
    have selfPointMem':" (∀i loc loc2. i < Toploc mem' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc
        (ShowL_nat i) ∧ accessStore loc mem' = Some (MPointer loc2) ⟶ loc = loc2)"
      using old unfolding ncpNewSelfPoint_def by blast
    then have old:"∀i l1 t1 l2 t2 loc i2 loc2.
     i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat i) ∧ LSubPrefL2
         loc2 (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc ⟶
     ¬ TypedMemSubPrefPtrs mem' l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs mem' l1 t1 loc loc2" using old
unfolding ncpElementsNoSubPref_def by blast

    show "ncpElementsNoSubPref (Memory st) m' " unfolding ncpElementsNoSubPref_def
    proof intros
      fix i l1 t1 l2 t2 loc i2 loc2
      assume c10:"i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat i) ∧
          LSubPrefL2 loc2 (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc"
      then have "accessStore loc (Memory st) = accessStore loc mem'"using sameLocs by auto
      have MConnew:"MCon (MTArray l1 t1) (Memory st) loc" using c10 by blast
      have locdef:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)" using c10 by simp

      then have inotl:"i < Toploc mem'" using b99 21(7) typeSafeDecl_hyps(5) ncp b98 c10 by simp
      then have inotlString:"(ShowL_nat i) ≠ l" using a1
        by (metis Read_Show_nat'_id linorder_neq_iff)
      have typeMemOld:"¬ TypedMemSubPrefPtrs mem' l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs mem' l1 t1
loc loc2" using old c10 MConnew by auto

      have "¬ LSubPrefL2 loc l" using c10 inotlString
        by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general a1 hash_inequality
hash_suffixes_associative)
      then have conc1:" ¬ TypedMemSubPrefPtrs m' l1 t1 loc loc2" using b105 b110 a110 MConnew a120
typeMemOld
        using inv_cpm2mTPrefOld_imps_TPref by blast
      then have hashlimit:"∀f. hash loc2 f ≠ loc" using c10
        by (metis (no_types, opaque_lifting) MCon_imps_Some LSubPrefL2_def Not_Sub_More_Specific
limitsOld not_Some_eq)

      have conc2:"¬ TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
      proof(cases " i2 < Toploc mem'")
        case True
        then have f2:" (∀p loc2. i2 < Toploc mem' ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc2
            (ShowL_nat i2) ∧ accessStore loc2 mem' = Some (MPointer p) ⟶ loc2 = p)"
```

```
          using selfPointMem' c10 by blast
        have "(ShowL_nat i2) ≠ (ShowL_nat i)" using c10
          by (metis hashesIntSame leD)

        then have loc2NotSub:"¬LSubPrefL2 loc2 l" using a1 True
          by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general c10 hash_inequality
hash_suffixes_associative hashesIntSame nat_neq_iff)
        show ?thesis
        proof
          assume asm:"TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
          then show False using hashlimit f2 c10 b110 loc2NotSub
          proof(induction t2 arbitrary:loc2 l2)
            case (MTArray x1 t2)
            then obtain i''' l''' where i'''def:"(i'''<l2 ∧ accessStore (hash loc2 (ShowL_nat
                i''')) m' = Some (MPointer l''') ∧ (l''' = loc ∨ TypedMemSubPrefPtrs m' x1 t2 l'''
loc))"
              using TypedMemSubPrefPtrs.simps(2)[of m' l2 x1 t2 loc2 loc] by blast
            then have "LSubPrefL2 (hash loc2 (ShowL_nat i''')) (ShowL_nat i2)" using MTArray(5)
              by (metis LSubPrefL2_def Not_Sub_More_Specific)
            moreover have "¬ LSubPrefL2 (hash loc2 (ShowL_nat i''')) l" using MTArray(7)
              using MemLSubPrefL2_specific_imps_general a1 by blast

            ultimately have l'''Exp:"l''' = (hash loc2 (ShowL_nat i'''))" using MTArray(4,6) i'''def
              by (metis True a110 c10)

            then show ?case
            proof(cases "l''' = loc")
              case True
              then show ?thesis using MTArray(3) l'''Exp by simp
            next
              case False
              then have "TypedMemSubPrefPtrs m' x1 t2 l''' loc" using i'''def by simp
              then show ?thesis using MTArray.IH[of x1 l'''] l'''Exp MTArray(3)
                using <LSubPrefL2 (hash loc2 (ShowL_nat
                    i''')) (ShowL_nat i2)> <¬ LSubPrefL2 (hash loc2 (ShowL_nat i''')) l> b110 c10 f2
hash_suffixes_associative by force
            qed
          next
            case (MTValue x)
            then have "(∃i<l2. hash loc2 (ShowL_nat i) = loc)" using TypedMemSubPrefPtrs.simps(1)[of m'
l2 x loc2 loc] by auto
            then show ?case using MTValue by simp
          qed
        qed
      next
        case False
        then show ?thesis
        proof(cases "i2 = Toploc mem'")
          case True
          then have g1:"(ShowL_nat i2) = l" using a1 by blast
          show ?thesis
          proof
            assume asm:" TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
            then show False using hashlimit c10 b105 selfPoint sameaccess
            proof(induction t2 arbitrary:loc2 l2)
              case (MTArray x1 t2)
              then obtain iIn lIn where idef: " (iIn<l2 ∧ accessStore (hash loc2 (ShowL_nat iIn)) m' =
Some
                  (MPointer lIn) ∧ (lIn = loc ∨ TypedMemSubPrefPtrs m' x1 t2 lIn loc))"
                using TypedMemSubPrefPtrs.simps(2)[of m' l2 x1 t2 loc2 loc] by blast
              have g2:"∀i. (hash loc2 (ShowL_nat i)) ≠ l" using g1 MTArray(4) LSubPrefL2_def
                by (metis hash_inequality hash_suffixes_associative)
              show ?case
              proof(cases "TypedMemSubPref (hash loc2 (ShowL_nat iIn)) l (MTArray x t)")
```

```
                case True
                then have g4:"lIn = (hash loc2 (ShowL_nat iIn))" using MTArray idef
                  using a110 by blast
                then show ?thesis
                proof(cases "lIn = loc")
                  case True
                  then show ?thesis using g4 MTArray by blast
                next
                  case False
                  have tp: "TypedMemSubPrefPtrs m' x1 t2 lIn loc" using False idef by simp
                  have sub: "LSubPrefL2 lIn l" using MTArray.prems(3) g1 g4 LSubPrefL2_hash_mono by
metis
                  have hl: "∀f. hash lIn f ≠ loc"
                    using sub <¬ LSubPrefL2 loc l> LSubPrefL2_hash LSubPrefL2_trans by metis
                  have c: "i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat
i) ∧
                      LSubPrefL2 lIn (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc"
                    using MTArray.prems(3) sub g1 by simp
                  show ?thesis using MTArray.IH[of x1 lIn] tp hl c MTArray.prems(4,5,6) by blast
                qed
              next
                case False
                then have " accessStore (hash loc2 (ShowL_nat iIn)) mem' = accessStore (hash loc2
(ShowL_nat iIn)) mm'" using g2 MTArray by blast
                moreover have "LSubPrefL2 (hash loc2 (ShowL_nat iIn)) l" using g1 MTArray
LSubPrefL2_def
                  by (metis hash_suffixes_associative)
                ultimately have "accessStore (hash loc2 (ShowL_nat iIn)) mm' = None" using MTArray by
metis
                then show ?thesis using idef
                  by (simp add: a110)
              qed
            next
              case (MTValue x')
              then show ?case by auto
            qed
          qed
        next
          case f4:False
          then have "i2> Toploc mem'" using False by auto
          then have loc2NotSub:"¬LSubPrefL2 loc2 l" using a1 c10
            by (metis LSubPrefL2_def f4 hash_suffixes_associative hashesInts)
          show ?thesis
          proof
            assume asm:"TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
            show False
            proof(cases t2)
              case (MTArray x11 x12)
              then have h1:"(∃i<l2. ∃l. accessStore (hash loc2 (ShowL_nat i)) m' = Some (MPointer l) ∧
(l = loc ∨ TypedMemSubPrefPtrs m' x11 x12 l loc))"
                using TypedMemSubPrefPtrs.simps(2)[of m' l2 x11 x12 loc2 loc] asm by blast
              then obtain i''' l''' where i'''def:"i'''<l2 ∧ accessStore (hash loc2 (ShowL_nat
                  i''')) m' = Some (MPointer l''') ∧ (l''' = loc ∨ TypedMemSubPrefPtrs m' x11 x12 l'''
loc)" by blast
              then have " accessStore (hash loc2 (ShowL_nat i''')) m' = None" using loc2NotSub
                by (metis False LSubPrefL2_def MemLSubPrefL2_specific_imps_general a1 a110 b110 c10
Not_Sub_More_Specific leI locationscd')
              then show ?thesis using i'''def by simp
            next
              case (MTValue x2)
              then have "(∃i<l2. hash loc2 (ShowL_nat i) = loc)"
                using asm TypedMemSubPrefPtrs.simps(1)[of m' l2 x2 loc2 loc] by blast
              then show ?thesis using hashlimit by simp
            qed
```

```
        qed

      qed
    qed
    then show "¬ TypedMemSubPrefPtrs m' l2 t2 loc2 loc" using inv_cpm2mTPrefOld_imps_TPref conc2 by
blast
    show "¬ TypedMemSubPrefPtrs m' l1 t1 loc loc2" using conc1 by simp
  qed
 next
  have old:"SomeValSomeTyp mem'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by blast
  have non:"∀ loc. LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶ accessStore loc mem' = None"
    using 21(7) typeSafeDecl_hyps(3) unfolding TypeSafe_def lessThanTopLocs_def
    by auto

  have somesomeT:"∀ destl'.
       TypedMemSubPref destl' (ShowL_nat (Toploc mem')) (MTArray x t) ⟶
       (∃ t. accessStore destl' m' = Some t) = (∃ tt. accessTypeStore destl' m' = Some tt)"
    using a3 unfolding cpm2m_def using cpm2m_TypeCompChange_somesome[of p l t _ c' x mm'] a4 a1 by
blast
  show "SomeValSomeTyp m'" unfolding SomeValSomeTyp_def
  proof intros
    fix locs
    show "(∃ t. accessStore locs m' = Some t) = (∃ tt. accessTypeStore locs m' = Some tt) "
    proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)")
      case True
      then show ?thesis using somesomeT by simp
    next
      case False
      then have "accessStore locs m' = accessStore locs mem'"
        using "21"(7)
        by (metis ‹∀ locations. accessStore locations c' = accessStore locations mem'› a1 a110 a3
cpm2mSingleChange_typedPref cpm2m_def)
      moreover have "accessTypeStore locs m' = accessTypeStore locs mem'"
        using False cpm2mSingleChange_typedPref_typed 21(7) allocateTypeSameAccess
        by (metis a1 a110 a3 aloc cpm2m_def)
      ultimately show ?thesis using old
        by (simp add: SomeValSomeTyp_def)
    qed
  qed

 next
  assume notCP:"¬cp"
  have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
cdc c' x mm'] b2 by simp
  have "Toploc (Memory st) ≤ Toploc mem'" using typeSafeDecl_hyps(5) 21(7) notCP by simp
  moreover have "Toploc c' > Toploc mem'" using a2 unfolding allocate_def by simp
  then show " Toploc (Memory st) ≤ Toploc m' " using b99 notCP typeSafeDecl_hyps 21 by simp
 next
  assume notCP:"¬cp"
  show "ncpNewSelfPoint (Memory st) m'" unfolding ncpNewSelfPoint_def
  proof intros
    fix i loc loc2
    assume asm:"i < Toploc m' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc (ShowL_nat i) ∧ accessStore
loc m' = Some (MPointer loc2)"
    have old:"(∀ i loc loc2. i < Toploc mem' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc (ShowL_nat
        i) ∧ accessStore loc mem' = Some (MPointer loc2) ⟶ loc = loc2)"
      using typeSafeDecl_hyps(5) notCP 21(7) unfolding ncpNewSelfPoint_def by blast
    have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l
t cdc c' x mm'] b2 by simp

    have a110:"m' = mm'" using a4 by simp
    then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
      using a3 a1 cpm2m_def[of p l x t cdc c'] cpm2mSingleChange[of p l t cdc c' x mm'] by fastforce
    have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc mem' = None" using locationscd' a2
```

```
lNotInC' a1 by auto
      have "∀locations. accessStore locations c' = accessStore locations mem'" using a2
accessAllocate[of c' mem'] by auto
      then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem' = accessStore locs mm'"
        using b100 a2 unfolding allocate_def by simp

      have g1:"Toploc c' = Suc (Toploc mem')" using a2 unfolding allocate_def by simp
      show "loc = loc2"
      proof(cases "i < Toploc mem'")
        case True
        then have "¬ LSubPrefL2 loc l" using a1 LSubPrefL2_def
          by (metis MemLSubPrefL2_specific_imps_general Read_Show_nat'_id asm hash_inequality
hash_suffixes_associative less_not_refl)
        then show ?thesis using old asm b110 a1 True a110 by auto
      next
        case False
        then have iIsTop:"i = Toploc mem'" using asm g1 b99 by simp
        have selfPoint:"∀la l'. TypedMemSubPref la l (MTArray x t) ∧ accessStore la mm' = Some
(MPointer l') ⟶ l' = la"
          using cpm2mSelfPointers[of p l t cdc c' x mm'] a3 a1 cpm2m_def[of p l x t cdc c'] by argo

        have sameaccess:"∀locs. locs ≠ l ∧ ¬ TypedMemSubPref locs l (MTArray x t) ⟶ accessStore
locs c' = accessStore locs mm'"
          using cpm2mSingleChange_typedPref[of p l t cdc c' x mm'] a3 a1 cpm2m_def[of p l x t cdc c']
by fastforce
        then have sameaccess:"∀locs. locs ≠ l ∧ ¬ TypedMemSubPref locs l (MTArray x t) ⟶
accessStore locs mem' = accessStore locs mm'"
          using b100 a2 allocate_def accessAllocate[of c' mem'] by simp
        then show ?thesis using asm iIsTop
          by (metis a1 a110 a3 cpm2m_iter_dest_unchanged b105 cpm2m_def hash_inequality lNotInC'
not_Some_eq selfPoint)
      qed
    qed
  next
    have b99:"Toploc c' = Toploc m'" using a3 a4 cpm2m_def[of p l x t cdc c' ] cpm2mTopLocSame[of p l t
cdc c' x mm'] b2 by simp
    then show "Toploc mem' ≤ Toploc m' " using aloc unfolding allocate_def by auto
  qed
qed
```

A memory array initialised from a storage pointer: the storage array is copied into fresh memory with *cps2m*.

```
lemma typeSafeDecl_22:
  assumes "tp = type.Memory (MTArray x t)"
    and "Some (v, t'') = Some (KStoptr p, type.Storage (STArray x' t'))"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = s"
    and "(cd', mem', sck', ev') = (cdc', mem''', sck'', ev'')"
    and "Some (c, m', k', e) = (if cps2mTypeCompatible (STArray x' t') (MTArray x t) ∧ Denvalue
        ev'' $$ ip = None then cps2m p (ShowL_nat (Toploc mem''')) x' t' s (snd (allocate
        mem''')) ⤜ (λm''. Some (cdc',
        m'', astack_dup ip (type.Memory (MTArray x t)) (KMemptr (ShowL_nat (Toploc mem''')), sck'',
ev'')) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 22 = assms
  then obtain l where a1:"l = ShowL_nat(Toploc mem''')" by simp
  then have locationscd':"(∀tloc loc. Toploc mem''' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc mem''' = None)"
    using typeSafeDecl_hyps(3) 22 unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have b1:"accessStore l mem''' = None" using a1 using Read_Show_nat'_id using LSubPrefL2_def by
auto
```

have *b2:"0 < x"* **using** *22(8) a1 cps2mTypeCompatible.simps(2)[of x' t' x t]*
by *fastforce*
then have *b3:"0 < x'"* **using** *22(8)* **by** *(auto split:if_split_asm)*
have *b4:"x = x'"* **using** *22(8)* **by** *(auto split:if_split_asm)*
then have *Mconcd:" SCon (STArray x' t') (extractValueType v) (Storage st (Address ev))"* **using** *typeSafeDecl_hyps(2) 22(2) 22(4) 22(5)* **by** *simp*
then have *MConcd2:" SCon (STArray x' t') p s"* **using** *22* **by** *simp*
obtain *c'* **where** *a2:"∃dud. (dud, c') = allocate mem'''"* **by** *(simp add: allocate_def)*
then have *aloc:"snd (allocate mem''') = c'"* **by** *(simp add: allocate_def)*
then have *lNotInC':"accessStore l c' = None"* **using** *b1 a1* **unfolding** *allocate_def accessStore_def* **by** *auto*
obtain *mm'* **where** *a3:"cps2m p l x' t' s c' = Some mm'"* **using** *22 a1 a2 typeSafeDecl_hyps*
using *b2 aloc b3* **by** *(metis (no_types, lifting) bind_eq_Some_conv option.discI)*
have *NoneIp:"Denvalue ev'' $$ ip = None"* **using** *22(8)* **by** *(auto split:if_split_asm)*
then have *a4:"Some (c, m', k', e) = Some (cdc', mm', astack_dup ip (type.Memory (MTArray x t)) (KMemptr l) (sck'', ev''))"*
using *a1 a2 a3 22 typeSafeDecl_hyps decl.simps(9) b2 aloc b3 NoneIp*
by *(metis bind.bind_lunit not_None_eq)*
then have *a4:"Some (c, m', k', e) = Some (cdc', mm', astack_dup ip (type.Memory (MTArray x t)) (KMemptr l) (sck', ev''))"*
using *22(7)* **by** *simp*
then have *a20:"k' = push (KMemptr l) sck'"* **using** *22(7) NoneIp* **by** *force*
have *a30:"e = (updateEnv ip (type.Memory (MTArray x t)) (Stackloc (ShowL$_{nat}$ (Toploc sck'))) ev'')"* **using** *a4 NoneIp* **by** *simp*
then have *a40:"Denvalue e = Denvalue(ev'' (| Denvalue := fmupd ip (type.Memory (MTArray x t),(Stackloc (ShowL$_{nat}$ (Toploc sck')))) (Denvalue ev'') |))"* **by** *simp*
then have *a50:"(Denvalue e) $$ ip = Some (type.Memory (MTArray x t),(Stackloc (ShowL$_{nat}$ (Toploc sck'))))"* **by** *simp*

have *a120:"lessThanTopLocs sck'"* **using** *typeSafeDecl_hyps(3) TypeSafe_def 22(7)* **by** *simp*
then have *a130:"∀v'''. accessStore (ShowL$_{nat}$ (Toploc sck')) sck' ≠ Some v'''"*
using *lessThanTopLocs_def Read_Show_nat'_id* **using** *LSubPrefL2_def* **by** *auto*
then have *a140:"∀x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL$_{nat}$ (Toploc sck'))))"*
using *TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 22(7)* **by** *fastforce*
then have *a150: "∀ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL$_{nat}$ (Toploc sck')))) ⟶ ip''' = ip"*
using *a40 lessThanTopLocs_def fmranI* **by** *auto*
then have *a160:"∀ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''')*
∧ l''' ≠ (Stackloc (ShowL$_{nat}$ (Toploc sck'))) ⟶ ip''' ≠ ip"
using *a40 lessThanTopLocs_def fmranI* **by** *auto*
have *cps2mTypeComp:"cps2mTypeCompatible (STArray x' t') (MTArray x t)"* **using** *22(8) a1* **by** *(meson option.discI)*
have *a110:"m' = mm'"* **using** *a4* **by** *simp*
then have *b27:"∀locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStore locs mem' = accessStore locs m'"*
using *a3 a1 cps2m_def[of p l x' t' s c'] cps2mSingleChange[of p l t' s c' x' mm'] a2 22(7)* **unfolding** *allocate_def*
by *(metis (no_types, lifting) Pair_inject allocateSameAccess aloc)*
have *b27T:"∀locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessTypeStore locs c' = accessTypeStore locs mm'"*
using *a3 a1 cps2m_def[of p l x' t' s c'] cps2mSingleChange_both[of p l t' s c' x' mm'] a2 22(7)* **unfolding** *allocate_def*
by *(metis (no_types, lifting))*
then have *b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"*
using *b27 a3 a1 cps2m_def[of p l x' t' " s" c'] cps2mSingleChange[of p l t' "s" c' x' mm']* **by** *fastforce*
have *b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem''' = None"* **using** *locationscd' a2 lNotInC' a1* **by** *auto*
have *"∀locations. accessStore locations c' = accessStore locations mem'''"* **using** *a2 accessAllocate[of c' mem''']* **by** *auto*
then have *b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem''' = accessStore locs mm'"*
using *b100 a2* **unfolding** *allocate_def* **by** *simp*
have *subMapping:"Mapping mem' ⊆$_f$ Mapping mm'"*

```
  proof -
    have "fmlookup (Mapping mem') k = Some v ⟹ fmlookup (Mapping mm') k = Some v" for k v
      using b105[rule_format, of k] b110[rule_format, of k] 22(7) unfolding accessStore_def
      by (cases "LSubPrefL2 k l") auto
    then show ?thesis unfolding fmsubset_alt_def fmpred_iff by blast
  qed
  have mcTop1:"MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
    using cps2m[of p l t' "s" c' x' mm' ] b3 Mconcd 22(2) b2
      cps2m_def[of p l x' t' "s" c'] b4 extractValueType.simps(4) "22"(6) 22(7)
    using MConcd2 cps2mTypeComp lNotInC' by presburger
  then have mcTop2:"MCon (MTArray x t) m' l" using a4 a110 22(7) by blast

  have "∀locations. accessStore locations c' = accessStore locations mem'" using a2 22(7)
accessAllocate[of c' mem'] by auto

  then have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                          ⟶ accessTypeStore locs c' = accessTypeStore locs m'"
    using cps2mSingleChange_both[of ] allocateTypeSameAccess a3 a1 a110 unfolding cps2m_def using 22(7)
a110 by blast
  moreover have "∀locs. accessTypeStore locs c' = accessTypeStore locs mem'" using aloc
allocateTypeSameAccess 22(7)
    by (metis prod.inject)
  ultimately have nonLocChanged2_Typed:"∀t' locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mem')) ∨ locs =
(ShowL_nat (Toploc mem'))
                                        ⟶ accessTypeStore locs mem' = accessTypeStore locs m'" by
simp
  have "∀destl'.
       TypedMemSubPref destl' l (MTArray x' t) ⟶
       (∃st. CompMemType mm' x' t st l destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) mm' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' mm' = Some (MTValue pval)))"
    using a1 a3 cps2mTypeComp unfolding cps2m_def using cps2m_TypeCompChange[of p l t' s c' x' mm' t]
by fastforce
  then have mtp:"∀destl'.
       TypedMemSubPref destl' (ShowL_nat (Toploc mem')) (MTArray x' t) ⟶
       (∃st. CompMemType m' x' t st (ShowL_nat (Toploc mem')) destl' ∧
             (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m' = Some parent_arr
              | MTValue pval ⇒ accessTypeStore destl' m' = Some (MTValue pval)))" using a110 22(7) a1
by blast
  have selfPoint:"∀la l'. TypedStoSubpref la l (STArray x' t') ∧ accessStore la mm' = Some (MPointer
l') ⟶ l' = la"
    using cps2mSelfPointers[of p l t' "s" c' x' mm'] a3 a1 cps2m_def[of p l x' t' s c']
    by (metis <∀locations. accessStore locations c' = accessStore locations mem'> a110 b27 lNotInC'
        option.discI)
  then have selfPoint2:"∀la l'. TypedMemSubPref la l (MTArray x t) ∧ accessStore la m' = Some
(MPointer l') ⟶ l' = la"
    using cps2mTypeComp a110
    using compatible_TypedStoSubpref_imps_TypedMemSubPref by presburger
  show ?thesis unfolding TypeSafe_def
  proof(intros)

    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 22(7) updateEnvUniqueLocs a4
NoneIp astack_dup_is_astack by fastforce
  next
    have b2: "compPointers sck' (Denvalue ev'')" using typeSafeDecl_hyps(3) 22(7) unfolding
TypeSafe_def by auto
    have "Address ev'' = Address e"using a30 by simp
    moreover have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 22(7) by simp
    ultimately show "compPointers k' (Denvalue e)"
      using compPointersNonStackUpd[of sck' ev'' e ip " type.Memory (MTArray x t)" k' "KMemptr l"]
        a40 a50 typeSafeDecl_hyps(3) 22(7) b2 a20 by simp
```

```
  next
    show "safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using a30 22(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def by
simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 22(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe a20 22(7) by metis
  next
    have "addressFormat (Address ev'')" using typeSafeDecl_hyps(3) TypeSafe_def 22(7) by simp
    then show "addressFormat(Address e)" using 22(7) a4 NoneIp by auto
  next
    have "addressFormat (Sender ev'')" using typeSafeDecl_hyps(3) TypeSafe_def 22(7) by simp
    then show "addressFormat (Sender e)" using 22(7) a4 NoneIp by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c" unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev'')) = (Storage st (Address e))" using a30 by simp

      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by
(simp add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = KMemptr l " using a170 True by simp
            have a190:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KMemptr l)" using a20 by (simp
add:push_def allocate_def updateStore_def accessStore_def)
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have a200:"tDen = type.Memory (MTArray x t)" using a90 a50 by simp
            have a220:"v = KStoptr p" using 22(2) by simp
            have a230:"cps2mTypeCompatible (STArray x' t') (MTArray x t)" using 22(8) a1 by (meson
option.discI)
            have "c = cdc'" using a4 by simp
            moreover have " (∃p. accessStore l c' = Some (MPointer p)) ∨ accessStore l c' = None"
              by (simp add: lNotInC')
            ultimately have "MCon (MTArray x t) mm' l" using a3 unfolding cpm2m_def
              using cps2m[of p l t' "s" c' x' mm' ] b3 Mconcd 22(2) b2
                cps2m_def[of p l x' t' "s" c'] b4 extractValueType.simps(4) "22"(6) 22(7) a100 by
(metis a220 a230)
            then have "MCon (MTArray x t) m' l" using a4 a110 22(7) by blast
            then show ?thesis using Stackloc some a180 a200 extractValueType.simps(4) 22(2) by simp
          qed
        next
          case False
          then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
            using a3 a1 cps2m_def[of p l x' t' " s" c'] cps2mSingleChange[of p l t' "s" c' x' mm'] by
fastforce

          have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem''' = None" using locationscd' a2
lNotInC' a1 by auto
```

```
        have "∀locations. accessStore locations c' = accessStore locations mem'''" using a2
accessAllocate[of c' mem'''] by auto
        then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem''' = accessStore locs
mm'"
          using b100 a2 unfolding allocate_def by simp

        then have "ip'' ≠ ip" using a160 a90 Stackloc False by simp
        then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Stackloc fmranI a90 False
* by fastforce
        then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 22 by blast
        then have a190:"accessStore x1 k' = accessStore x1 sck'"
          using a20 False by (simp add:push_def allocate_def updateStore_def accessStore_def)
        then show ?thesis
        proof(cases "accessStore x1 k'")
          case None
          then show ?thesis using a180 a190 Stackloc by simp
        next
          case some:(Some a)
          then show ?thesis
          proof(cases a)
            case (KValue x1)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a100 22(7) by (cases tDen; fastforce)
          next
            case (KCDptr x2)
            then have b10:"(case tDen of Calldata struct ⇒ MCon struct cdc' x2 | _ ⇒ False)"
              using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 22(7) unfolding TypeSafe_def
typeCompat_def by force
            then obtain struct where tden: "tDen = Calldata struct" by (auto split:type.splits)
            then have "MCon struct cdc' x2" using b10 by simp
            then show ?thesis using Stackloc some KCDptr tden a110 a4 by simp
          next
            case (KMemptr x3)
            then have b10:"(case tDen of type.Memory struct ⇒ MCon struct mem' x3 | _ ⇒ False)"
              using some a170 Stackloc a190 using typeSafeDecl_hyps(3) 22(7) unfolding TypeSafe_def
typeCompat_def by force
            then obtain struct where tden: "tDen = type.Memory struct" by (auto split:type.splits)
            then have "MCon struct mem''' x3" using b10 22 by simp
            then have "MCon struct mm' x3" using mconCopySingle[of l mem''' mm' struct] b110 b105 by
auto
            then show ?thesis using Stackloc some KMemptr tden a110 a4 by simp
          next
            case (KStoptr x4)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a100 22(7) by (cases tDen; fastforce)
          qed
        qed
      qed
    next
      case (Storeloc x2)
      then have "ip'' ≠ ip" using a160 a90 by simp
      then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev'')" using a40 Storeloc fmranI a90 * by
fastforce
      then have a180:"(case tDen of type.Storage typ ⇒ SCon typ x2 (Storage st (Address ev'')) | _
⇒ False)"
        using a170 Storeloc using typeSafeDecl_hyps(3) 22(7)
        unfolding TypeSafe_def typeCompat_def by force
      then obtain typ' where tdent:"tDen = type.Storage typ'" by (auto split: type.splits)
      then have "SCon typ' x2 (Storage st (Address ev''))" using a180 by simp
      then have "SCon typ' x2 (Storage st (Address e))" using a100 by simp
      then show ?thesis using a170 Storeloc using typeSafeDecl_hyps(3) 22(7) tdent
        by simp
    qed
```

```
    qed
  next
    show "lessThanTopLocs c" using typeSafeDecl_hyps(3) 22(7) a4 unfolding TypeSafe_def by auto
  next
    have b100:"lessThanTopLocs mem'''" using typeSafeDecl_hyps(3) 22(7) unfolding TypeSafe_def by
blast
    have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' "s" c'] cps2mTopLocSame[of p l
t' "s" c' x' mm'] b3 by simp
    have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs m'"
      using cps2mSingleChange[of p l t' "s" c' x' mm'] a4 a3 cps2m_def[of p l x' t' "s" c'] by
fastforce
    moreover have "∀locations. accessStore locations c' = accessStore locations mem'''" using a2
accessAllocate[of c' mem'''] 22 by auto
    ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs m' = accessStore locs mem'''"
by simp
    show "lessThanTopLocs m'" using cdMemLocsToploc[of mem''' l c' m'] b100 a1 a2 b99 using b10 by
presburger
  next
    show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
  next
    fix locs tp assume ncp:"¬cp"
    have a110:"m' = mm'" using a4 by simp
    then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
      using a3 a1 cps2m_def[of p l x' t' s c'] cps2mSingleChange[of p l t' "s" c' x' mm'] by fastforce
    have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem''' = None" using locationscd' a2
lNotInC' a1 by auto
    have "∀locations. accessStore locations c' = accessStore locations mem'''" using a2
accessAllocate[of c' mem'''] by auto
    then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem''' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def by simp
    have a120:"∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs" using typeSafeDecl_hyps(5)
22(7) ncp by simp
    moreover have "∀locs tp. MCon tp mem''' locs ⟶ MCon tp mm' locs" using mconCopySingle[of l
mem''' mm' ] using b110 b105 by simp
    moreover have "Toploc c' > Toploc mem''''" using a2 unfolding allocate_def by simp
    moreover have b99:"Toploc c' = Toploc mm'" using a1 a3 a4 cps2m_def[of p l x' t' s c']
cps2mTopLocSame[of p l t' "s" c' x' mm'] b2 by fastforce
    ultimately show "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using typeSafeDecl_hyps(5) 22(7)
a110 a120 ncp by blast
  next
    assume notCP:"¬cp"
    show "ncpDenvalueLimit e ev k' (Stack st) (Memory st)" unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p''' i''
      assume asm:" (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' =
Some (KMemptr p''') ∧ i'' <
          Toploc (Memory st) ∧ LSubPrefL2 p''' (ShowL_nat i'')"
      then obtain i where idef:"Denvalue e $$ i = Some (type.Memory tp', Stackloc locs)" by blast

      have "ncpDenvalueLimit ev'' ev sck' (Stack st) (Memory st)" using 22 notCP typeSafeDecl_hyps(5)
by blast
      then have old:"∀tp' locs p i.
     (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'') ∧ accessStore locs sck' = Some (KMemptr
p) ∧ i < Toploc
         (Memory st) ∧ LSubPrefL2 p (ShowL_nat i) ⟶
     (∃tp'' loc2 p'.
         (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
         accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧ tp'' = tp' ∨ (∃len arr. p' ≠ p ∧
tp''
             = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p)))"
        unfolding ncpDenvalueLimit_def by blast
      show "∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p''' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
```

```
p'''
            ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p'''))"
      proof(cases "locs = (ShowL_nat (Toploc sck'))")
        case True
        have b50:"p''' = l" using a40 a20 push_def
          by (metis stackvalue.inject(3) True accessStore_def accessStore_updateStore allocateMapping
asm old.prod.exhaust option.inject snd_eqD)
        then have " p''' = (ShowL_nat (Toploc mem'''))" using a1 by blast
        moreover have "Toploc mem''' ≥ Toploc (Memory st)" using 22 notCP typeSafeDecl_hyps(5) by
blast
        ultimately show ?thesis using asm
          by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general hash_inequality
hash_suffixes_associative hashesIntSame le_antisym nat_less_le)
      next
        case False
        then have b5:"locs ≠ (ShowL_nat (Toploc sck'))" using a20 a40 asm
          by (metis)
        then have "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev'')" using a40 a160 fmranI
asm by fastforce
        moreover have "accessStore locs sck' = accessStore locs k'" using b5 a20 unfolding push_def
accessStore_def updateStore_def allocate_def by auto
        ultimately show ?thesis using old asm by metis
      qed
    qed
  next
    assume ncp:"¬cp"
    show "ncpOMemInDMem (Memory st) m'" unfolding ncpOMemInDMem_def
    proof intros
      fix i loc
      assume c10:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)"
      then have " accessStore loc (Memory st) = accessStore loc mem'''" using typeSafeDecl_hyps(5)
22(7) ncpOMemInDMem_def ncp by auto
      have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs m'"
        using cps2mSingleChange[of p l t' "s" c' x' mm'] a4 a3 cps2m_def[of p l x' t' s c'] by
fastforce
      moreover have "∀locations. accessStore locations c' = accessStore locations mem''''" using a2
accessAllocate[of c' mem'''] by auto
      ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs m' = accessStore locs
mem''''" by simp
      have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' s c'] cps2mTopLocSame[of p l
t' "s" c' x' mm'] b2
        by fastforce
      have "Toploc (Memory st) ≤ Toploc mem''''" using typeSafeDecl_hyps(5) c10 22(7) ncpOMemInDMem_def
ncp by simp
      then have "(ShowL_nat i) ≠ l" using a1 typeSafeDecl_hyps(5) c10 b99
        by (metis hashesIntSame leD)
      then show "accessStore loc (Memory st) = accessStore loc m'" using c10 a1 b99 a2
cdMemLocsToploc[of mem''' l c' m']
        using "22"(3)
        by (metis LSubPrefL2_def ‹accessStore loc (Memory st) = accessStore loc mem'''› b10
hash_suffixes_associative hashesInts)
      have " ∀locs. ¬ LSubPrefL2 locs l ⟶ accessTypeStore locs c' = accessTypeStore locs m'"
        using cps2mSingleChange_both[of p l t' "s" c' x' mm'] a4 a3 cps2m_def[of p l x' t' s c'] by
fastforce
      moreover have "∀locations. accessTypeStore locations c' = accessTypeStore locations mem''''"
using a2 allocateTypeSameAccess
        by (metis aloc)
      ultimately have b10:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessTypeStore locs m' = accessTypeStore
locs mem''''" by simp
      moreover have " accessTypeStore loc (Memory st) = accessTypeStore loc mem''''" using
typeSafeDecl_hyps(5) 22(7) ncpOMemInDMem_def ncp c10 by blast
      ultimately show "accessTypeStore loc (Memory st) = accessTypeStore loc m'"
        using c10 a1 b99 a2 cdMemLocsToploc[of mem''' l c' m'] 22(3) b10 a2
        by (metis LSubPrefL2_def MemLSubPrefTransitive ‹(ShowL_nat i) ≠ l› hash_flatten_right
```

```
hash_inequality)
    qed
  next
    assume notCP:"¬cp"
    have a110:"m' = mm'" using a4 by simp
    then have b100:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
      using a3 a1 cps2m_def[of p l x' t' s c'] cps2mSingleChange[of p l t' "s" c' x' mm'] by fastforce
    have b105:"∀loc. LSubPrefL2 loc l ⟶ accessStore loc mem''' = None" using locationscd' a2
lNotInC' a1 by auto
    have "∀locations. accessStore locations c' = accessStore locations mem'''" using a2
accessAllocate[of c' mem'''] by auto
    then have b110:"∀locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem''' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def by simp

    fix locs tp
    have a120:"∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem''' locs" using typeSafeDecl_hyps(5)
22(7) notCP by simp
    moreover have "∀locs tp. MCon tp mem''' locs ⟶ MCon tp mm' locs" using mconCopySingle[of l
mem''' mm' ] using b110 b105 by simp
    moreover have b98:"Toploc c' > Toploc mem'''" using a2 unfolding allocate_def by simp
    moreover have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' s c']
cps2mTopLocSame[of p l t' "s" c' x' mm'] b2 by fastforce
    ultimately have "MCon tp (Memory st) locs ⟹ MCon tp m' locs" using 22 typeSafeDecl_hyps(5) 22(7)
a110 a120 notCP by blast


    have sameaccess:" ∀locs. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x' t') ⟶ accessStore locs
c' = accessStore locs mm'"
      using cps2mSingleChange_typedPref[of p l t' "s" c' x' mm'] a3 a1 cps2m_def[of p l x' t' s c'] by
fastforce
    then have sameaccess:"∀locs. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x' t') ⟶ accessStore
locs mem''' = accessStore locs mm'"
      using b100 a2 unfolding allocate_def a2 accessAllocate[of c' mem''']
      by (simp add: <∀locations. accessStore locations c' = accessStore locations mem'''>)

    have limitsOld:" ((∀tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None) ∧
     (∀loc y. accessStore loc (Memory st) = Some y ⟶ (∃tloc<Toploc (Memory st). LSubPrefL2 loc
(ShowL_nat tloc))))"
      using typeSafeDecl_hyps(1) unfolding TypeSafe_def lessThanTopLocs_def using 22(7) by auto

    have limits:"((∀tloc loc. Toploc mem''' ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore
loc mem''' = None) ∧
                  (∀loc y. accessStore loc mem''' = Some y ⟶ (∃tloc<Toploc mem'''. LSubPrefL2 loc
(ShowL_nat tloc))))"
      using typeSafeDecl_hyps(3) 22(7) unfolding TypeSafe_def lessThanTopLocs_def by simp

    have old:"ncpElementsNoSubPref (Memory st) mem''' ∧ ncpOMemInDMem (Memory st) mem''' ∧
ncpNewSelfPoint (Memory st) mem''''" using 22 notCP typeSafeDecl_hyps(5) by blast
    then have sameLocs:"(∀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
       accessStore loc (Memory st) = accessStore loc mem''')" unfolding ncpOMemInDMem_def by blast
    have selfPointMem':" (∀i loc loc2. i < Toploc
       mem''' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc (ShowL_nat i) ∧ accessStore loc mem''' =
       Some (MPointer loc2) ⟶ loc = loc2)"
      using old unfolding ncpNewSelfPoint_def by blast
    then have old:"∀i l1 t1 l2 t2 loc i2 loc2.
     i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat i) ∧ LSubPrefL2
        loc2 (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc ⟶
     ¬ TypedMemSubPrefPtrs mem''' l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs mem''' l1 t1 loc loc2" using
old unfolding ncpElementsNoSubPref_def by blast

    show "ncpElementsNoSubPref (Memory st) m' " unfolding ncpElementsNoSubPref_def
    proof intros
      fix i l1 t1 l2 t2 loc i2 loc2
```

```
      assume c10:"i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat i) ∧
          LSubPrefL2 loc2 (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc"
      then have "accessStore loc (Memory st) = accessStore loc mem'''' "using sameLocs by auto
      have MConnew:"MCon (MTArray l1 t1) (Memory st) loc" using c10 by blast
      have locdef:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)" using c10 by simp

      then have inotl:"i < Toploc mem'''" using b99 22(7) typeSafeDecl_hyps(5) notCP b98 c10 by simp
      then have inotlString:"(ShowL_nat i) ≠ l" using a1 22(7)
        by (metis Read_Show_nat'_id linorder_neq_iff)
      have typeMemOld:"¬ TypedMemSubPrefPtrs mem''' l2 t2 loc2 loc ∧ ¬ TypedMemSubPrefPtrs mem''' l1
t1 loc loc2" using old c10 MConnew by auto

      have "¬ LSubPrefL2 loc l" using c10 inotlString
        by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general a1 hash_inequality
hash_suffixes_associative)
      then have conc1:" ¬ TypedMemSubPrefPtrs m' l1 t1 loc loc2" using b105 b110 a110 MConnew a120
typeMemOld
        using inv_cpm2mTPrefOld_imps_TPref by blast
      then have hashlimit:"∀f. hash loc2 f ≠ loc" using c10
        by (metis (no_types, opaque_lifting) MCon_imps_Some LSubPrefL2_def Not_Sub_More_Specific
limitsOld not_Some_eq)

      have conc2:"¬ TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
      proof(cases " i2 < Toploc mem'''")
        case True
        then have f2:" (∀p loc2. i2 < Toploc
            mem''' ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc2 (ShowL_nat i2) ∧ accessStore loc2 mem'''
= Some (MPointer p) ⟶ loc2 = p)"
          using selfPointMem' c10 by blast
        have "(ShowL_nat i2) ≠ (ShowL_nat i)" using c10
          by (metis hashesIntSame leD)

        then have loc2NotSub:"¬LSubPrefL2 loc2 l" using a1 True
          by (metis LSubPrefL2_def MemLSubPrefL2_specific_imps_general c10 hash_inequality
hash_suffixes_associative hashesIntSame nat_neq_iff)
        show ?thesis
        proof
          assume asm:"TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
          then show False using hashlimit f2 c10 b110 loc2NotSub
          proof(induction t2 arbitrary:loc2 l2)
            case (MTArray x1 t2)
            then obtain i''' l''' where i'''def:"(i'''<l2 ∧ accessStore (hash loc2 (ShowL_nat
                i''')) m' = Some (MPointer l''') ∧ (l''' = loc ∨ TypedMemSubPrefPtrs m' x1 t2 l'''
loc))"
              using TypedMemSubPrefPtrs.simps(2)[of m' l2 x1 t2 loc2 loc] by blast
            then have "LSubPrefL2 (hash loc2 (ShowL_nat i''')) (ShowL_nat i2)" using MTArray(5)
              by (metis LSubPrefL2_def Not_Sub_More_Specific)
            moreover have "¬ LSubPrefL2 (hash loc2 (ShowL_nat i''')) l" using MTArray(7)
              using MemLSubPrefL2_specific_imps_general a1 by blast

            ultimately have l'''Exp:"l''' = (hash loc2 (ShowL_nat i'''))" using MTArray(4,6) i'''def
              by (metis True a110 c10)

            then show ?case
            proof(cases "l''' = loc")
              case True
              then show ?thesis using MTArray(3) l'''Exp by simp
            next
              case False
              then have "TypedMemSubPrefPtrs m' x1 t2 l''' loc" using i'''def by simp
              then show ?thesis using MTArray.IH[of x1 l'''] l'''Exp MTArray(3)
                using <LSubPrefL2 (hash loc2 (ShowL_nat
                    i''')) (ShowL_nat i2)> <¬ LSubPrefL2 (hash loc2 (ShowL_nat i''')) l> b110 c10 f2
hash_suffixes_associative by force
```

```
          qed
        next
          case (MTValue x)
          then have "(∃i<l2. hash loc2 (ShowL_nat i) = loc)" using TypedMemSubPrefPtrs.simps(1)[of m'
l2 x loc2 loc] by auto
          then show ?case using MTValue by simp
        qed
      qed
    next
      case False
      then show ?thesis
      proof(cases "i2 = Toploc mem'''")
        case True
        then have g1:"(ShowL_nat i2) = l" using a1 by blast
        show ?thesis
        proof
          assume asm:" TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
          then show False using hashlimit c10 b105 selfPoint sameaccess
          proof(induction t2 arbitrary:loc2 l2)
            case (MTArray x1 t2)
            then obtain iIn lIn where idef: " (iIn<l2 ∧ accessStore (hash loc2 (ShowL_nat iIn)) m' =
Some
                (MPointer lIn) ∧ (lIn = loc ∨ TypedMemSubPrefPtrs m' x1 t2 lIn loc))"
              using TypedMemSubPrefPtrs.simps(2)[of m' l2 x1 t2 loc2 loc] by blast
            have g2:"∀i. (hash loc2 (ShowL_nat i)) ≠ l" using g1 MTArray(4) LSubPrefL2_def
              by (metis hash_inequality hash_suffixes_associative)
            show ?case
            proof(cases "TypedStoSubpref (hash loc2 (ShowL_nat iIn)) l (STArray x' t')")
              case True
              then have g4:"lIn = (hash loc2 (ShowL_nat iIn))" using MTArray idef
                using a110 by blast
              then show ?thesis
              proof(cases "lIn = loc")
                case True
                then show ?thesis using g4 MTArray by blast
              next
                case False
                have tp: "TypedMemSubPrefPtrs m' x1 t2 lIn loc" using False idef by simp
                have sub: "LSubPrefL2 lIn l" using MTArray.prems(3) g1 g4 LSubPrefL2_hash_mono by
metis
                have hl: "∀f. hash lIn f ≠ loc"
                  using sub <¬ LSubPrefL2 loc l> LSubPrefL2_hash LSubPrefL2_trans by metis
                have c: "i < Toploc (Memory st) ∧ Toploc (Memory st) ≤ i2 ∧ LSubPrefL2 loc (ShowL_nat
i) ∧
                    LSubPrefL2 lIn (ShowL_nat i2) ∧ MCon (MTArray l1 t1) (Memory st) loc"
                  using MTArray.prems(3) sub g1 by simp
                show ?thesis using MTArray.IH[of x1 lIn] tp hl c MTArray.prems(4,5,6) by blast
              qed
            next
              case False
              then have " accessStore (hash loc2 (ShowL_nat iIn)) mem''' = accessStore (hash loc2
(ShowL_nat iIn)) mm'" using g2 MTArray by blast
              moreover have "LSubPrefL2 (hash loc2 (ShowL_nat iIn)) l" using g1 MTArray
LSubPrefL2_def
                by (metis hash_suffixes_associative)
              ultimately have "accessStore (hash loc2 (ShowL_nat iIn)) mm' = None" using MTArray by
metis
              then show ?thesis using idef
                by (simp add: a110)
            qed
          next
            case (MTValue x')
            then show ?case by auto
          qed
```

```
        qed
      next
        case f4:False
        then have "i2> Toploc mem'''" using False by auto
        then have loc2NotSub:"¬LSubPrefL2 loc2 l" using a1 c10
          by (metis LSubPrefL2_def f4 hash_suffixes_associative hashesInts)
        show ?thesis
        proof
          assume asm:"TypedMemSubPrefPtrs m' l2 t2 loc2 loc"
          show False
          proof(cases t2)
            case (MTArray x11 x12)
            then have h1:"(∃ i<l2. ∃ l. accessStore (hash loc2 (ShowL_nat i)) m' = Some (MPointer l) ∧
(l = loc ∨ TypedMemSubPrefPtrs m' x11 x12 l loc))"
              using TypedMemSubPrefPtrs.simps(2)[of m' l2 x11 x12 loc2 loc] asm by blast
            then obtain i''' l''' where i'''def:"i'''<l2 ∧ accessStore (hash loc2 (ShowL_nat
                i''')) m' = Some (MPointer l''') ∧ (l''' = loc ∨ TypedMemSubPrefPtrs m' x11 x12 l'''
loc)" by blast
            then have " accessStore (hash loc2 (ShowL_nat i''')) m' = None" using loc2NotSub
              by (metis False LSubPrefL2_def MemLSubPrefL2_specific_imps_general a1 a110 b110 c10
Not_Sub_More_Specific leI locationscd')
            then show ?thesis using i'''def by simp
          next
            case (MTValue x2)
            then have "(∃ i<l2. hash loc2 (ShowL_nat i) = loc)"
              using asm TypedMemSubPrefPtrs.simps(1)[of m' l2 x2 loc2 loc] by blast
            then show ?thesis using hashlimit by simp
          qed
        qed
      qed
      then show "¬ TypedMemSubPrefPtrs m' l2 t2 loc2 loc" using inv_cpm2mTPrefOld_imps_TPref conc2 by
blast
      show " ¬ TypedMemSubPrefPtrs m' l1 t1 loc loc2" using conc1 by simp
    qed
  next
    have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' s c'] cps2mTopLocSame[of p l t'
s c' x' mm'] b2 by simp
    show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t2 l ptr_loc
      assume *:" (type.Memory t2, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KMemptr ptr_loc)"

      show "case t2 of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)

       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
      proof(cases "(type.Memory t2, Stackloc l) |∈| fmran(Denvalue ev')")
        case True
        moreover have sameACC:"accessStore l k' = accessStore l sck'"
          using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
          by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
              fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
        then have old:"(case t2 of
                        MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' =
Some arr)
                        | MTValue val ⇒ accessTypeStore ptr_loc mem' = Some (MTValue val))"
          using typeSafeDecl_hyps(1) 22(7) * True TypeSafe_def typeSafeDecl_hyps(3)
denvalueTypeCorrectness_def by fastforce
        have mcOld:"MCon t2 mem' ptr_loc" using * sameACC True typeSafeDecl_hyps(1) 22(7)
          unfolding TypeSafe_def typeCompat_def
          by (metis typeSafeDecl_hyps(3) sameMemTSafe)
```

```
        have lims:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) ptr_loc"
          using typeSafeAllPtrsNotTop[OF typeSafeDecl_hyps(3) True] * 22(7) sameACC b99 aloc unfolding
allocate_def
          using LSubPrefL2_def MemLSubPrefTransitive by force
        then show ?thesis
        proof(cases t2)
          case (MTArray x11 x12)
          have nonLocChanged2_Typed_cond:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶
accessTypeStore loc m' = accessTypeStore loc mem'"
            using nonLocChanged2_Typed by simp
          have nonLochanged:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶ accessStore loc m' =
accessStore loc mem'"
            using b27 a1 a110 22(7) by auto
          have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) mem' = Some x12) "
            using old MTArray by simp

          show ?thesis
            using MTArray old' nonLocChanged2_Typed_cond lims
              22(7) nonLochanged MemLSubPrefL2_specific_imps_general mtypes.simps(5) old

            by (metis (no_types, lifting) ext)
        next
          case (MTValue x2)
          then show ?thesis using nonLocChanged2_Typed old 22(7) lims by simp
        qed
      next
        case False
        then have "Denvalue e $$ ip = Some (type.Memory t2, Stackloc l)"
          using a40 * a4 typeSafeDecl_hyps(4) decl_env_not_i fmlookup_ran_iff by fast
        then have t2Def:"l = (ShowL_nat (Toploc sck')) ∧ t2 = (MTArray x t)" using a40 by simp
        then have acc:"accessStore l k' = Some (KMemptr (ShowL_nat (Toploc mem')))"
          using a40 a20 a1 * 22(7) unfolding push_def allocate_def updateStore_def accessStore_def by
auto
        then have ptrEQ:"ptr_loc = (ShowL_nat (Toploc mem'))" using * by auto
        have conc1:"(∀ i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some t)"
          using a3 unfolding cps2m_def using cps2m_TypeCompChangeIndexs[of p "(ShowL_nat (Toploc
mem'''))" t' s c' x' mm' t]
            cps2mTypeComp 22(7) "*" a110 a1 acc b4 by auto
        have "accessStore ptr_loc m' = accessStore ptr_loc mem'"
          using 22(7) ptrEQ t2Def "*" a1 a110 acc b1 b27 by simp
        moreover have "accessStore ptr_loc mem' = None"
          using 22(7) ptrEQ a1 typeSafeDecl_hyps(3) unfolding TypeSafe_def lessThanTopLocs_def
          by (meson LSubPrefL2_def order_eq_refl)
        ultimately have " (∀ v. accessStore ptr_loc m' = Some (MPointer v) ⟶ accessTypeStore ptr_loc
m' = Some (MTArray x t))"
          by simp
        then show ?thesis using t2Def conc1 by simp
      qed
    qed
  next
    show "subPrefixStructuralConsistency m'" unfolding subPrefixStructuralConsistency_def
    proof(intros)
      fix locs tp
      assume in1:"accessTypeStore locs m' = Some tp"

      have "∀ locations. accessStore locations c' = accessStore locations mem'" using a2 22(7)
accessAllocate[of c' mem'] by auto

      have nonLocChanged_Typed:"∀ locs. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t')
                          ⟶accessTypeStore locs mem' = accessTypeStore locs m'"
        using a3 a1 unfolding cps2m_def using aloc cps2mSingleChange_typedPref_both[of p l t' s c' x
mm'] allocateTypeSameAccess a2 a110
```

```
    by (simp add: ‹∀locs. accessTypeStore locs c' = accessTypeStore locs mem'› b4)
  then have nonLocChanged_Typed:"∀locs. locs ≠ l ∧ ¬ TypedMemSubPref locs l (MTArray x t)
                    ⟶accessTypeStore locs mem' = accessTypeStore locs m'" using cps2mTypeComp
    using b4 compatible_TypedStoSubpref_imps_TypedMemSubPref_neg by presburger

  have "MCon (MTArray x t) mm' l"
    using mcTop1 by blast
  then have mcTop:"MCon (MTArray x t) m' l" using a110 by blast

  have prefPtrs_imps_Pref:"∀x3. TypedMemSubPrefPtrs mm' x t l x3 ⟶ TypedMemSubPref x3 l (MTArray
x t)"
    using selfPoint_imps_TypedMemSubPref[OF ] selfPoint compatible_TypedStoSubpref_imps_TypedMemSubPref_neg
cps2mTypeComp a110 by blast

  show "case accessStore locs m' of None ⇒ False | Some (MValue v) ⇒ ∃val. MCon tp m' locs ∧ tp
= MTValue val ∧ accessTypeStore locs m' = Some tp
   | Some (MPointer p) ⇒ ∃len arr. MCon tp m' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore
(hash p (ShowL_nat i)) m' = Some arr)
                    "
  proof(cases "accessTypeStore locs mem' = Some tp")
    case True
    then have accL:"∃v. accessStore locs mem' = Some v"
      using typeSafeDecl_hyps(3) 22(7) unfolding TypeSafe_def SomeValSomeTyp_def by simp
    have notSub:"¬ LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
    proof
      assume c:"LSubPrefL2 locs (ShowL_nat (Toploc mem'))"
      then have "accessStore locs mem' = None"
        using 22(7) nless_le a1 b105 by simp
      then show False using accL by auto
    qed
    then have sameACC:" accessStore locs m' = accessStore locs mem'" using b27 22(7) a1 by simp
    then consider (ptr) p where "accessStore locs m' = Some (MPointer p)"
      | (val) val2 where "accessStore locs m' = Some (MValue val2)"
      using typeSafeDecl_hyps(1) 22(7) True unfolding TypeSafe_def subPrefixStructuralConsistency_def
      by (metis memoryvalue.exhaust accL)
    then show ?thesis
    proof(cases)
      case ptr
      then obtain len arr where
        old:"MCon tp mem' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat
i)) mem' = Some arr)
              "
        using typeSafeDecl_hyps(3) 22(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
      then have lim2:"∀len arr loc. tp = MTArray len arr ∧ TypedMemSubPrefPtrs mem' len arr p loc
                  ⟶ ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc
mem')) loc"
        using AccessedMemPtrsCantTop[of "mem'" tp p] typeSafeDecl_hyps(3) 22(7) unfolding
TypeSafe_def by blast
      have lim:" ¬ LSubPrefL2 p (ShowL_nat (Toploc mem')) ∧ ¬ LSubPrefL2 (ShowL_nat (Toploc mem'))
p"
        using AllPtrsNotTop_MCon notSub ptr using typeSafeDecl_hyps(3) 22(7) old unfolding
TypeSafe_def by blast
      have accTLoc:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc mem')) ∨ loc =
          (ShowL_nat (Toploc mem')) ⟶ accessStore loc m' = accessStore loc mem'"
        using 22(7) a1 a110 b27 by simp
      have "MCon tp m' p" using MCon_memory_transfer[OF _ lim lim2 accTLoc] old by blast
      moreover have "∀i<len. ¬ LSubPrefL2 (hash p (ShowL_nat i)) (ShowL_nat (Toploc mem'))" using
lim
        using MemLSubPrefL2_specific_imps_general by blast
      ultimately have " MCon tp m' p ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore (hash p
(ShowL_nat i)) m' = Some arr)"
        using nonLocChanged2_Typed old 22(7) by simp
      then show ?thesis using ptr by auto
```

```
        next
          case val
          then have old:"∃val. MCon tp mem' locs ∧ tp = MTValue val ∧ accessTypeStore locs mem' =
Some tp"
            using typeSafeDecl_hyps(3) 22(7) True sameACC unfolding TypeSafe_def
subPrefixStructuralConsistency_def by fastforce
          then have "∃val. MCon tp m' locs ∧ tp = MTValue val ∧ accessTypeStore locs m' = Some tp"
            using nonLocChanged2_Typed MCon.simps val notSub 22(7) sameACC by fastforce
          then show ?thesis using val by auto
        qed
      next
        case False
        have "locs ≠ (ShowL_nat (Toploc mem'))"
          using False in1 nonLocChanged2_Typed by fastforce
        then have sub:"TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)"
          using nonLocChanged_Typed a1 22(7) in1 False by auto

        then have "TypedMemSubPref locs (ShowL_nat (Toploc mem')) (MTArray x t)"
          using selfPoint_imps_TypedMemSubPref by simp
        then obtain st where stD:"(CompMemType m' x t st (ShowL_nat (Toploc mem')) locs ∧
           (case st of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash locs
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore locs m' = Some (MTValue pval)))" using mtp a1
          using b4 by blast
        then have cmp:"CompMemType m' x t st (ShowL_nat (Toploc mem')) locs " by blast

        have mcLocs:"MCon st m' locs" using cmp mcTop CompTypeRemainsMCon a1 22(7) by blast

        show ?thesis
        proof(cases st)
          case (MTArray x11 x12)
          have subs:"t = MTArray x11 x12 ∧ (∃i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat
i)) m' = Some (MPointer locs)) ∨
                       (∃midP subL subA i. CompMemType m' x t (MTArray subL subA) (ShowL_nat (Toploc
mem')) midP
                       ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧
subA = MTArray x11 x12)"
            using CompMemType_imps_Mid[of m' x t x11 x12 "(ShowL_nat (Toploc mem'))" locs] cmp MTArray
by simp

          then have acc:"accessStore locs m' = Some (MPointer locs)"
            using selfPoint2 mcLocs sub MTArray MCon_imps_TypedMemSubPref_Some[OF _ _ selfPoint2, of
locs] mcTop a1 a110 22(7)
            by (metis False MCon.simps(2) <∀locs. accessTypeStore locs c' = accessTypeStore locs mem'>
b27T in1 nonLocChanged_Typed option.distinct(1))

          moreover have "∃len arr. MCon tp m' locs ∧ tp = MTArray len arr ∧ (∀i<len. accessTypeStore
(hash locs (ShowL_nat i)) m' = Some arr)
                                ∧ (∀v. accessStore locs m' = Some (MPointer v) ⟶ accessTypeStore locs
m' = Some (MTArray len arr))"
            using in1
          proof(cases "(∃i<x. accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some
(MPointer locs) ∧ t = MTArray x11 x12)")
            case True
            then obtain i where idef:"i<x ∧ accessStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))
m' = Some (MPointer locs)
                                      ∧ t = MTArray x11 x12"
              by auto
            then have "TypedMemSubPref (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) (ShowL_nat (Toploc
mem')) (MTArray x t) " by auto
            then have locsD:"locs = (hash (ShowL_nat (Toploc mem')) (ShowL_nat i))" using selfPoint2
idef a1 a110 22(7) by blast

            have "∀i<x. accessTypeStore (hash l (ShowL_nat i)) mm' = Some t"
```

```
            using a3 unfolding cps2m_def
            using a110 allocateTypeSameAccess cps2m_TypeCompChangeIndexs[of p l t' s c' x mm' t]
cps2mTypeComp b4 by blast
          then have "∀i<x. accessTypeStore (hash (ShowL_nat (Toploc mem')) (ShowL_nat i)) m' = Some
(MTArray x11 x12)"
            using idef a1 22(7) a110 by blast
          then have "tp = MTArray x11 x12" using in1 locsD idef by simp
          moreover have "(∀i<x11. accessTypeStore (hash locs (ShowL_nat i)) m' = Some x12)" using
stD MTArray by simp
          ultimately show ?thesis using mcLocs MTArray in1 by blast
        next
          case False
          then obtain midP i subL subA where mid:"(CompMemType m' x t (MTArray subL subA) (ShowL_nat
(Toploc mem')) midP
                ∧ accessStore (hash midP (ShowL_nat i)) m' = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
            using subs by blast
          then have locD:"locs = (hash midP (ShowL_nat i))" using selfPoint2 a110 a1 22(7)
            using CompMemType_imps_in_GetAllMemoryLocations_ptr mcTop memSet_selfPoint by blast
          then obtain st2 where st2D:"(CompMemType m' x t st2 (ShowL_nat (Toploc mem')) midP ∧
           (case st2 of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash midP
(ShowL_nat i)) m' = Some parent_arr
            | MTValue pval ⇒ accessTypeStore midP m' = Some (MTValue pval)))" using mid mtp a1 a110
22(7)
            using CompMemTypeSameLocsSameType CompMemType_imps_TypedMemSubPrefPtrs prefPtrs_imps_Pref
b4 by blast
          then have "st2 = (MTArray subL subA)" using mid CompMemTypeSameLocsSameType mcTop a1 22(7)
by blast
          then have " ∀i<subL. accessTypeStore (hash midP (ShowL_nat i)) m' = Some subA" using st2D
by simp
          then have "MTArray x11 x12 = tp" using in1 locD mid by simp
          then show ?thesis using mcLocs MTArray stD in1 by fastforce
        qed
        ultimately show ?thesis by auto
      next
        case (MTValue x2)
        then obtain v where vDef:"accessStore locs m' = Some (MValue v)"
          using mcLocs MCon.simps(1)[of x2 m' locs] by (auto split:option.splits memoryvalue.splits)
        moreover have "accessTypeStore locs m' = Some (MTValue x2)" using stD MTValue by auto
        moreover have "tp = MTValue x2" using calculation in1 by simp
        moreover have "MCon tp m' locs" using mcLocs calculation MTValue by blast
        ultimately show ?thesis using MTValue by simp
      qed
    qed
  qed
next

  have old:"SomeValSomeTyp mem'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by blast
  have non:"∀loc. LSubPrefL2 loc (ShowL_nat (Toploc mem')) ⟶ accessStore loc mem' = None"
    using 22(7) typeSafeDecl_hyps(3) unfolding TypeSafe_def lessThanTopLocs_def
    by auto

  have somesomeT:"∀destl'. TypedMemSubPref destl' l (MTArray x t) ⟶ (∃t. accessStore destl' m' =
Some t) = (∃tt. accessTypeStore destl' m' = Some tt)"
    using a3 unfolding cps2m_def using cps2m_TypeCompChange_somesome[of p l t' s c' x mm' t] a4 a1
cps2mTypeComp 22(7) by fastforce
  show "SomeValSomeTyp m'" unfolding SomeValSomeTyp_def
  proof intros
    fix locs
    show "(∃t. accessStore locs m' = Some t) = (∃tt. accessTypeStore locs m' = Some tt) "
    proof(cases "TypedMemSubPref locs l (MTArray x t)")
      case True
      then show ?thesis using somesomeT by simp
    next
```

```
        case False
        show ?thesis
        proof (cases "locs = l")
          case True
          have cc0:"accessStore locs m' = accessStore locs mem'"
            using b27 a110 True
            by presburger
          have cc1:"accessTypeStore locs m' = accessTypeStore locs mem'"
            using nonLocChanged2_Typed a1 True a110
            using ‹∀ locs. accessTypeStore locs c' = accessTypeStore locs mem'› b27T by presburger
          then show ?thesis using old SomeValSomeTyp_def cc0 by auto
        next
          case locs_neq: False
          have nsub_mem:"¬ TypedMemSubPref locs l (MTArray x t)"
            using False by simp
          have nsub_sto:"¬ TypedStoSubpref locs l (STArray x' t')"
          proof
            assume sto:"TypedStoSubpref locs l (STArray x' t')"
            then have "TypedMemSubPref locs l (MTArray x t) ∨ locs = l"
              using cps2mTypeComp compatible_TypedStoSubpref_imps_TypedMemSubPref by blast
            then show False using nsub_mem locs_neq by blast
          qed
          have cc0:"accessStore locs m' = accessStore locs mem'"
            using "22"(7) a1 a110 a3 a2 ‹∀ locations. accessStore locations c' = accessStore locations
mem'› locs_neq nsub_sto
            unfolding cps2m_def using cps2mSingleChange_typedPref[of p l t' s c' x' mm'] by simp
          have cc1:"accessTypeStore locs m' = accessTypeStore locs mem'"
            using "22"(7) a1 a110 a3 a2 ‹∀ locations. accessTypeStore locations c' = accessTypeStore
locations mem'› locs_neq nsub_sto
            unfolding cps2m_def using cps2mSingleChange_typedPref_both[of p l t' s c' x' mm'] by simp
          then show ?thesis using old SomeValSomeTyp_def cc0 by auto
        qed
      qed
    qed

  next
    assume ncp:"¬cp"
    have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' s c'] cps2mTopLocSame[of p l t'
"s" c' x' mm'] b2 by fastforce
    have "Toploc (Memory st) ≤ Toploc mem'''" using typeSafeDecl_hyps(5) 22(7) ncp by simp
    moreover have "Toploc c' > Toploc mem'''" using a2 unfolding allocate_def by simp
    then show " Toploc (Memory st) ≤ Toploc m' " using b99 ncp typeSafeDecl_hyps 22 by simp
  next
    assume notCP:"¬cp"
    show "ncpNewSelfPoint (Memory st) m'" unfolding ncpNewSelfPoint_def
    proof intros
      fix i loc loc2
      assume asm:"i < Toploc m' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc (ShowL_nat i) ∧ accessStore
loc m' = Some (MPointer loc2)"
      have old:"(∀ i loc loc2. i < Toploc
          mem''' ∧ Toploc (Memory st) ≤ i ∧ LSubPrefL2 loc (ShowL_nat i) ∧ accessStore loc mem''' =
          Some (MPointer loc2) ⟶ loc = loc2)"
        using typeSafeDecl_hyps(5) notCP 22(7) unfolding ncpNewSelfPoint_def by blast
      have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' s c'] cps2mTopLocSame[of p l
t' "s" c' x' mm'] b2 by fastforce

      have a110:"m' = mm'" using a4 by simp
      then have b100:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs c' = accessStore locs mm'"
        using a3 a1 cps2m_def cps2mSingleChange[of p l t' "s" c' x' mm'] by fastforce
      have b105:"∀ loc. LSubPrefL2 loc l ⟶ accessStore loc mem''' = None" using locationscd' a2
lNotInC' a1 by auto
      have "∀ locations. accessStore locations c' = accessStore locations mem'''" using a2
accessAllocate[of c' mem'''] by auto
      then have b110:"∀ locs. ¬ LSubPrefL2 locs l ⟶ accessStore locs mem''' = accessStore locs mm'"
```

```
        using b100 a2 unfolding allocate_def by simp

      have g1:"Toploc c' = Suc (Toploc mem''')" using a2 unfolding allocate_def by simp
      show "loc = loc2"
      proof(cases "i < Toploc mem'''")
        case True
        then have "¬ LSubPrefL2 loc l" using a1 LSubPrefL2_def
          by (metis MemLSubPrefL2_specific_imps_general Read_Show_nat'_id asm hash_inequality
hash_suffixes_associative less_not_refl)
        then show ?thesis using old asm b110 a1 True a110 by auto
      next
        case False
        then have iIsTop:"i = Toploc mem'''" using asm g1 b99 by simp
        have selfPoint:"∀ la l'. la ≠ l ∧ TypedStoSubpref la l (STArray x' t') ∧ accessStore la mm' =
Some (MPointer l') ⟶ l' = la"
          using cps2mSelfPointers[of p l t' "s" c' x' mm'] a3 a1 cps2m_def
          by auto

        have sameaccess:"∀ locs. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x' t') ⟶ accessStore
locs c' = accessStore locs mm'"
          using cps2mSingleChange_typedPref[of p l t' "s" c' x' mm'] a3 a1 cps2m_def by fastforce
        then have sameaccess:"∀ locs. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x' t') ⟶
accessStore locs mem''' = accessStore locs mm'"
          using b100 a2 allocate_def accessAllocate[of c' mem'''] by simp
        then show ?thesis using asm iIsTop
          by (metis a1 a110 a3 b105 cps2mAccessPrePost cps2m_def hash_inequality lNotInC' not_Some_eq
selfPoint)
      qed
    qed
  next
    have b99:"Toploc c' = Toploc m'" using a3 a4 cps2m_def[of p l x' t' "s" c'] cps2mTopLocSame[of p l
t' "s" c' x' mm'] b3 by simp
    then show "Toploc mem' ≤ Toploc m' " using aloc 22 unfolding allocate_def by auto
  qed
qed
```

A storage array initialised from a storage pointer: the variable refers to the storage array.

```
lemma typeSafeDecl_42:
  assumes "tp = type.Storage (STArray x t)"
    and "Some (v, t'') = Some (KStoptr p, tv0)"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem'', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = type.Storage (STArray x t) ∧ Denvalue ev'' $$ ip = None then
Some (cdc', mem'', astack_dup ip
        (type.Storage (STArray x t)) (KStoptr p) (sck'', ev'')) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 42 = assms
  then have a10:"t'' = type.Storage (STArray x t)" using decl.simps(2) by (simp split:if_splits)
  have NoneIp:" Denvalue ev'' $$ ip = None" using 42 by (simp split:if_splits)
  then have a40:"Some (c, m', k', e) = Some (cdc', mem'', astack ip (type.Storage (STArray x t))
(KStoptr p) (sck', ev''))"
    using 42 decl.simps(2) a10 typeSafeDecl_hyps(1) by simp
  then have k'Def:"k' = push (KStoptr p) sck'" by force

  have eDef:"e = updateEnv ip (type.Storage (STArray x t)) (Stackloc (ShowL_nat (Toploc sck')))
ev''"using a40 unfolding astack.simps by blast
  then have eIP:"(Denvalue e) $$ ip = Some ((type.Storage (STArray x t)),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp
  have sameADD:"Address ev' = Address e"
```

```
    using typeSafeDecl_hyps(4) decl_env by blast

  have wxDef:"tv0 = type.Storage (STArray x t)" using 42(8) by (simp split:if_splits)

  have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
  then have a130:"∀v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
    using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
  then have a140:"∀x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
    using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 42(7) by fastforce
  then have a150: "∀ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip"
    using a40 lessThanTopLocs_def fmranI by auto

  obtain locO tpO pO where oldAccess:"(
        (type.Storage tpO, locO) |∈| fmran (Denvalue ev') ∧
        (case locO of
         Stackloc stloc ⇒
           accessStore stloc sck' = Some (KStoptr pO) ∧ (tpO = (STArray x t) ∧ v = KStoptr pO ∨
extractValueType v ≠
               pO ∧ CompStoType tpO (STArray x t) pO (extractValueType v))
         | Storeloc stloc ⇒ tpO = (STArray x t) ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧
CompStoType tpO (STArray x t) stloc (extractValueType v)))"
    using typeSafeDecl_hyps 42 wxDef by blast

  have compOld: " (∀tp1 tp2 l1 l2 l1' l2' stl1 stl2.
        (type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
        (type.Storage tp2, l2) |∈| fmran (Denvalue ev') ∧
        (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
        (l2 = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2) ⟶
        (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if TypedStoSubpref
stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
    using typeSafeDecl_hyps(3) 42 unfolding TypeSafe_def compPointers_def by blast
  then have compOld':"(∀tp1 l1 l1' l2' stl1 stl2.
        (type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
        (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
        (locO = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2) ∧ stl2 = pO ∨ locO = Storeloc
stl2) ⟶
        (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tpO stl1 stl2 else if TypedStoSubpref
stl1 stl2 tpO then CompStoType tpO tp1 stl2 stl1 else True))"
    using oldAccess by blast
  show ?thesis unfolding TypeSafe_def
  proof(intros)
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 42(7) updateEnvUniqueLocs a40 by
fastforce
  next
    have b2: "compPointers sck' (Denvalue ev')" using typeSafeDecl_hyps(3) 42(7) unfolding TypeSafe_def
by auto

    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    show "compPointers k' (Denvalue e)" unfolding compPointers_def
    proof intros
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume in1:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then obtain i1 i2 where iDef:" Denvalue e $$ i1 = Some (type.Storage tp1, l1) ∧ Denvalue e$$ i2
= Some (type.Storage tp2, l2)" by blast

      consider (bothStacks) "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1)
                  ∧ l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2)"
        | (StackLoc1) "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∧ l2 = Storeloc
stl2"
```

```
        | (StackLoc2) "l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∧ l1 = Storeloc
stl1"
        | (StoreLocs) "l1 = Storeloc stl1 ∧ l2 = Storeloc stl2" using in1 by auto

      then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases)
        case bothStacks
        then show ?thesis
        proof(cases "l1 = Stackloc (ShowL_nat (Toploc sck'))")
          case True
          then have stl1IsP:"stl1 = p" using in1 k'Def typeSafeDecl_hyps 42 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
          have "i1 = ip" using iDef eDef a150 True by blast
          then have tp1IsStArray:"tp1 = STArray x t" using iDef eDef by auto
          then show ?thesis
          proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
            case True
            then have stl2IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 42 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
            have "i2 = ip" using iDef eDef a150 True by blast
            then have tp2IsStArray:"tp2 = STArray x t" using iDef eDef by auto
            then show ?thesis using stl2IsP tp2IsStArray tp1IsStArray stl1IsP by auto
          next
            case False
            then have acc2:"accessStore l2' k' = accessStore l2' sck'"
              using k'Def bothStacks
              by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)
            then have i2NotIP:"i2 ≠ ip" using eDef iDef False by auto
            then have tp2InOld:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
              using fmranI iDef by fastforce
            have sameAcc2:" accessStore l2' sck' = Some (KStoptr stl2)" using acc2 in1 bothStacks by
auto
            then show ?thesis
            proof(cases locO)
              case (Stackloc x1)
              then have acc:"accessStore x1 sck' = Some (KStoptr pO)" using oldAccess by simp
              then have cc:"(tpO = STArray x t ∧ v = KStoptr pO ∨ extractValueType v ≠ pO ∧
CompStoType tpO (STArray x t) pO (extractValueType v))"
                using compOld oldAccess using 42(7) Stackloc by simp
              have comps:"(if TypedStoSubpref pO stl2 tp2 then CompStoType tp2 tpO stl2 pO else
                           if TypedStoSubpref stl2 pO tpO then CompStoType tpO tp2 pO stl2 else True)"
                using compOld' sameAcc2 bothStacks tp2InOld Stackloc 42(7) acc by blast
              then show ?thesis
              proof(cases "v = KStoptr pO")
                case True
                then have "tpO = STArray x t" using cc True by simp
                then have sames:"stl1 = pO ∧ tp1 = tpO" using tp1IsStArray True 42 stl1IsP in1 by
blast

                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tpO (STArray x t) pO stl1" using stl1IsP cc 42 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
              qed
            next
              case (Storeloc x2)
              then have cc:"(tpO = STArray x t ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧
CompStoType tpO (STArray x t) x2 (extractValueType v))"
```

```
              using compOld oldAccess using 42(7) by simp
            have comps:"(if TypedStoSubpref x2 stl2 tp2 then CompStoType tp2 tpO stl2 x2 else
                        if TypedStoSubpref stl2 x2 tpO then CompStoType tpO tp2 x2 stl2 else True)"
              using compOld' sameAcc2 bothStacks tp2InOld Storeloc 42(7) by blast
            then show ?thesis
            proof(cases "v = KStoptr x2")
              case True
              then have "tpO = STArray x t" using cc True by simp
              then have sames:"stl1 = x2∧ tp1 = tpO" using tp1IsStArray True 42 stl1IsP in1 by
blast

              then show ?thesis using compPointers_if_sym[OF comps] comps by simp
            next
              case False
              then have comps2:"CompStoType tpO (STArray x t) x2 stl1" using stl1IsP cc 42 by simp
              then show ?thesis using compPointers_if_child[OF comps comps2]
                compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
            qed
          qed
        qed
      next
        case False
        then have acc1:"accessStore l1' k' = accessStore l1' sck'"
          using k'Def bothStacks
          by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)

        then have i1NotIp:"i1 ≠ ip" using False eDef iDef by auto
        then have In''1:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
          using fmranI iDef by fastforce
        then show ?thesis
        proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
          case True
          then have stl2IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 42 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
          have "i2 = ip" using iDef eDef a150 True by blast
          then have tp2IsStArray:"tp2 = STArray x t" using iDef eDef by auto
          have tp1InOld:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev')" using In''1 42(7) by blast
          then show ?thesis
          proof(cases locO)
            case (Stackloc x1)
            then have acc:"accessStore x1 sck' = Some (KStoptr pO)" using oldAccess by simp
            then have cc:"(tpO = STArray x t ∧ v = KStoptr pO ∨ extractValueType v ≠ pO ∧
CompStoType tpO (STArray x t) pO (extractValueType v))"
              using compOld oldAccess using 42(7) Stackloc by simp
            have comps:"(if TypedStoSubpref pO stl1 tp1 then CompStoType tp1 tpO stl1 pO else
                        if TypedStoSubpref stl1 pO tpO then CompStoType tpO tp1 pO stl1 else True)"
              using compOld' acc1 in1 bothStacks tp1InOld Stackloc 42(7) acc by metis
            then show ?thesis
            proof(cases "v = KStoptr pO")
              case True
              then have "tpO = STArray x t" using cc True by simp
              then have sames:"stl2 = pO ∧ tp2 = tpO" using tp2IsStArray True 42 stl2IsP in1 by
blast

              then show ?thesis using compPointers_if_sym[OF comps] comps by simp

            next
              case False
              then have comps2:"CompStoType tpO (STArray x t) pO stl2" using stl2IsP cc 42 by simp

              then show ?thesis using compPointers_if_child[OF comps comps2]
                compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp2IsStArray by (simp
```

```
only:)
              qed
            next
              case (Storeloc x2)
              then have cc:"(tpO = STArray x t ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧
CompStoType tpO (STArray x t) x2 (extractValueType v))"
                using compOld oldAccess using 42(7) by simp
              have comps:"(if TypedStoSubpref x2 stl1 tp1 then CompStoType tp1 tpO stl1 x2 else
                           if TypedStoSubpref stl1 x2 tpO then CompStoType tpO tp1 x2 stl1 else True)"
                using compOld' acc1 in1 bothStacks tp1InOld Storeloc 42(7) by metis
              then show ?thesis
              proof(cases "v = KStoptr x2")
                case True
                then have "tpO = STArray x t" using cc True by simp
                then have sames:"stl2 = x2∧ tp2 = tpO" using tp2IsStArray True 42 stl2IsP in1 by
blast
                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tpO (STArray x t) x2 stl2" using stl2IsP cc 42 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp2IsStArray by (simp
only:)
              qed
            qed
          next
            case f2:False
            then have acc2:"accessStore l2' k' = accessStore l2' sck'"
              using k'Def bothStacks
              by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)
            then have i2NotIP:"i2 ≠ ip" using eDef iDef f2 by auto
            then have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
              using fmranI iDef by fastforce
            have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue ev') ∧
     (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1)) ∧
     (l2 = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2))" using 42(7) In''2 In''1
bothStacks acc1 acc2 in1 by auto
            then show ?thesis using b2 unfolding compPointers_def by blast
          qed
        qed
       next
        case StackLoc1
        then have "i2 ≠ ip" using iDef eDef by auto
        then have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
          using fmranI iDef by fastforce
        then show ?thesis
        proof(cases "l1 = Stackloc (ShowL_nat (Toploc sck'))")
          case True
          then have stl1IsP:"stl1 = p" using in1 k'Def typeSafeDecl_hyps 42 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
          have "i1 = ip" using iDef eDef a150 True by blast
          then have tp1IsStArray:"tp1 = STArray x t" using iDef eDef by auto
          then show ?thesis
          proof(cases locO)
            case (Stackloc x1)
            then have acc:"accessStore x1 sck' = Some (KStoptr pO)" using oldAccess by simp
            then have cc:"(tpO = STArray x t ∧ v = KStoptr pO ∨ extractValueType v ≠ pO ∧
CompStoType tpO (STArray x t) pO (extractValueType v))"
              using compOld oldAccess using 42(7) Stackloc by simp
            have comps:"(if TypedStoSubpref pO stl2 tp2 then CompStoType tp2 tpO stl2 pO else
                         if TypedStoSubpref stl2 pO tpO then CompStoType tpO tp2 pO stl2 else True)"
```

```
            using compOld' In''2 Stackloc 42(7) acc StackLoc1 by blast
          then show ?thesis
          proof(cases "v = KStoptr pO")
            case True
            then have "tpO = STArray x t" using cc True by simp
            then have sames:"stl1 = pO ∧ tp1 = tpO" using tp1IsStArray True 42 stl1IsP in1 by blast

            then show ?thesis using compPointers_if_sym[OF comps] comps by simp
          next
            case False
            then have comps2:"CompStoType tpO (STArray x t) pO stl1" using stl1IsP cc 42 by simp
            then show ?thesis using compPointers_if_child[OF comps comps2]
              compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
          qed
        next
          case (Storeloc x2)
          then have cc:"(tpO = STArray x t ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧ CompStoType
tpO (STArray x t) x2 (extractValueType v))"
            using compOld oldAccess using 42(7) by simp
          have comps:"(if TypedStoSubpref x2 stl2 tp2 then CompStoType tp2 tpO stl2 x2 else
                        if TypedStoSubpref stl2 x2 tpO then CompStoType tpO tp2 x2 stl2 else True)"
            using compOld' StackLoc1 In''2 Storeloc 42(7) by blast
          then show ?thesis
          proof(cases "v = KStoptr x2")
            case True
            then have "tpO = STArray x t" using cc True by simp
            then have sames:"stl1 = x2∧ tp1 = tpO" using tp1IsStArray True 42 stl1IsP in1 by blast

            then show ?thesis using compPointers_if_sym[OF comps] comps by simp
          next
            case False
            then have comps2:"CompStoType tpO (STArray x t) x2 stl1" using stl1IsP cc 42 by simp

            then show ?thesis using compPointers_if_child[OF comps comps2]
              compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
          qed
        qed
      next
        case False
        then have "i1 ≠ ip" using iDef eDef by auto
        then have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')"
          using eDef iDef fmranI by fastforce
        moreover have "accessStore l1' sck' = Some (KStoptr stl1)"
          using StackLoc1 k'Def False unfolding push_def allocate_def accessStore_def updateStore_def
by auto
        ultimately show ?thesis using compOld 42(7) StackLoc1 using In''2 by blast
      qed
    next
      case StackLoc2
      then have "i1 ≠ ip" using iDef eDef by auto
      then have In''2:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
        using fmranI iDef by fastforce
      then show ?thesis
      proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
        case True
        then have stl1IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 42 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
        have "i2 = ip" using iDef eDef a150 True by blast
        then have tp1IsStArray:"tp2 = STArray x t" using iDef eDef by auto
        then show ?thesis
        proof(cases locO)
          case (Stackloc x1)
```

```
            then have acc:"accessStore x1 sck' = Some (KStoptr pO)" using oldAccess by simp
            then have cc:"(tpO = STArray x t ∧ v = KStoptr pO ∨ extractValueType v ≠ pO ∧
CompStoType tpO (STArray x t) pO (extractValueType v))"
              using compOld oldAccess using 42(7) Stackloc by simp
            have comps:"(if TypedStoSubpref pO stl1 tp1 then CompStoType tp1 tpO stl1 pO else
                          if TypedStoSubpref stl1 pO tpO then CompStoType tpO tp1 pO stl1 else True)"
              using compOld' In''2 Stackloc 42(7) acc StackLoc2 by blast
            then show ?thesis
            proof(cases "v = KStoptr pO")
              case True

              then have "tpO = STArray x t" using cc True by simp
              then have sames:"stl2 = pO ∧ tp2 = tpO" using tp1IsStArray True 42 stl1IsP in1 by blast

              then show ?thesis using compPointers_if_sym[OF comps] comps by simp
            next
              case False
              then have comps2:"CompStoType tpO (STArray x t) pO stl2" using stl1IsP cc 42 by simp

              then show ?thesis using compPointers_if_child[OF comps comps2]
                compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
            qed
          next
            case (Storeloc x2)
            then have cc:"(tpO = STArray x t ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧ CompStoType
tpO (STArray x t) x2 (extractValueType v))"
              using compOld oldAccess using 42(7) by simp
            have comps:"(if TypedStoSubpref x2 stl1 tp1 then CompStoType tp1 tpO stl1 x2 else
                          if TypedStoSubpref stl1 x2 tpO then CompStoType tpO tp1 x2 stl1 else True)"
              using compOld' StackLoc2 In''2 Storeloc 42(7) by blast
            then show ?thesis
            proof(cases "v = KStoptr x2")
              case True

              then have "tpO = STArray x t" using cc True by simp
              then have sames:"stl2 = x2∧ tp2 = tpO" using tp1IsStArray True 42 stl1IsP in1 by simp

              then show ?thesis using compPointers_if_sym[OF comps] comps by simp
            next
              case False
              then have comps2:"CompStoType tpO (STArray x t) x2 stl2" using stl1IsP cc 42 by simp

              then show ?thesis using compPointers_if_child[OF comps comps2]
                compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
            qed
          qed
        next
          case False
          then have "i2 ≠ ip" using iDef eDef by auto
          then have "(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')"
            using eDef iDef fmranI by fastforce
          moreover have "accessStore l2' sck' = Some (KStoptr stl2)"
            using StackLoc2 k'Def False unfolding push_def allocate_def accessStore_def updateStore_def
by auto
          ultimately show ?thesis using compOld 42(7) StackLoc2 using In''2 by blast
        qed
      next
        case StoreLocs
        then have "i1 ≠ ip" using iDef eDef by auto
        moreover have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
          using fmranI iDef calculation by fastforce
        moreover have "i2 ≠ ip" using iDef eDef StoreLocs by auto
```

```
        moreover have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef calculation
          using fmranI iDef by fastforce
        ultimately show ?thesis using compOld StoreLocs 42(7) by blast
      qed
    qed
  next
    have *:"safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def
using 42(7) by auto
    have **:"Address ev' = Address e" using eDef 42(7) by simp
    have ***:"Contract ev' = Contract e" using eDef 42(7) by simp
    show "safeContract (Accounts st) (Storage st) " using * by auto
  next
    show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
  next
    show "svalueTypes (Svalue e)" using eDef 42(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def
by simp
  next
    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    then show "lessThanTopLocs k'" using stackPushToplocSafe k'Def 42(7) by metis
  next
    have "lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    then show "lessThanTopLocs c" using stackPushToplocSafe 42(7) a40 by simp
  next
    have "lessThanTopLocs mem''" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    then show "lessThanTopLocs m'" using 42(7) a40 by simp
  next
    have "addressFormat (Address ev')" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    then show "addressFormat(Address e)" using 42(7) a40 by auto
  next
    have "addressFormat (Sender ev')" using typeSafeDecl_hyps(3) TypeSafe_def 42(7) by simp
    then show "addressFormat (Sender e)" using 42(7) a40 by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c " unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev')) = (Storage st (Address e))" using
typeSafeDecl_hyps(3) eDef
        using sameADD by presburger
      have a110:"m' = mem''" and a115:"cdc' = c" using a40 by simp+

      then have a160:"∀ ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using eDef lessThanTopLocs_def fmranI by auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KStoptr p)"
            using k'Def by (simp add:push_def allocate_def updateStore_def accessStore_def)

          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = (KStoptr p) " using a170 True by simp
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have "tDen = type.Storage (STArray x t)" using a90 eDef by simp
            moreover have "(∀ locs tp. SCon tp locs (Storage st (Address ev)) ⟶ SCon tp locs (Storage
```

```
st (Address ev'))) "
              using typeSafeDecl_hyps(8) 42 wxDef by simp
            moreover have "SCon (STArray x t) p (Storage st (Address e))"
              using typeSafeDecl_hyps(2) 42(2) wxDef calculation sameADD by simp
            ultimately show ?thesis using Stackloc some a180 by simp
          qed
        next
          case False

          then have "ip'' ≠ ip" using a160 a90 Stackloc by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev')" using eDef Stackloc fmranI a90 False
* 42(7) by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 42(7) by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'" using k'Def False by (simp
add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a115 a100 42(7) by (cases a; cases tDen;
force+)
          qed
        qed
      next
        case (Storeloc x2)
        then have a170:"ip'' ≠ ip" using eDef Storeloc a90 by auto
        then have a180: "(tDen, lDen) |∈| fmran (Denvalue ev')" using eDef 42(7) Storeloc fmranI a90
by fastforce
        then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def using
Storeloc a100 42(7) by (cases tDen;force)
      qed
    qed
  next
    show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp

  next
    have a110:"m' = mem''" using a40 by simp
    then show "⋀locs tp. ¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m' locs" using
typeSafeDecl_hyps(5) 42(7) typeSafeDecl_hyps(5) by (simp)
  next
    assume "¬ cp"
    then show "Toploc (Memory st) ≤ Toploc m'" using typeSafeDecl_hyps(5) 42(7) a40 by blast
  next
    assume notCP:"¬cp"
    show "ncpDenvalueLimit e ev k' (Stack st) (Memory st) "
      unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p'' i
      assume a120:" (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' =
Some (KMemptr
          p'') ∧ i < Toploc (Memory st) ∧ LSubPrefL2 p'' (ShowL_nat i)"

      then have a160:"∀ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using eDef lessThanTopLocs_def fmranI by auto
      show " ∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p'' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'' ∧
              tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p''))
```

```
        "
      proof -
        obtain ip'' where ip''def:"(Denvalue e) $$ ip'' = Some (type.Memory tp', Stackloc locs)" using
a120 by blast
        then have a170:"ip'' ≠ ip" using eDef by auto
        then have a180: "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev')" using eDef 42(7)
fmranI ip''def by fastforce
        have a190:"locs ≠ (ShowL_nat (Toploc sck'))" using a160 a170 k'Def
          using a150 ip''def by blast

        then obtain y' where a200:"accessStore locs sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) 42(7) a180 by blast
        then have "accessStore locs k' = accessStore locs sck'" using k'Def a120 a190 unfolding
push_def allocate_def updateStore_def accessStore_def by simp
        then show " ∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p'' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'' ∧
              tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p''))"
          using typeSafeDecl_hyps(5) 42(7) a120 a40 eDef k'Def a120 a180 notCP ncpDenvalueLimit_def
          by (metis)
      qed
    qed
  next
    assume notCP:"¬cp"
    then show "ncpOMemInDMem (Memory st) m'" using a40 42(7) typeSafeDecl_hyps by simp
  next
    assume notCP:"¬cp"
    then show "ncpElementsNoSubPref (Memory st) m'" using a40 typeSafeDecl_hyps 42 by blast
  next
    assume notCP:"¬cp"
    then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 42 by blast
  next
    show "Toploc mem' ≤ Toploc m' " using a40 42 by simp
  next
    show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t l ptr_loc
      assume *:" (type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some (KMemptr
ptr_loc)"
      then have "(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev')"
        using a40 unfolding astack.simps updateEnv.simps using type.distinct(7) typeSafeDecl_hyps(4)
decl_env_not_i fmlookup_ran_iff
          option.inject prod.inject
        by fastforce
      moreover have "accessStore l k' = accessStore l sck'"
        using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
        by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
            fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
      ultimately show "case t of
       MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
        using a40 typeSafeDecl_hyps(3) * unfolding TypeSafe_def denvalueTypeCorrectness_def using 42(7)
by auto
    qed
  next
    show "subPrefixStructuralConsistency m'"
      using typeSafeDecl_hyps(3) a40 42(7) unfolding TypeSafe_def subPrefixStructuralConsistency_def by
auto
  next
    show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a40 42(7) by
blast
  qed
```

qed

A storage mapping initialised from a storage pointer: the variable refers to the mapping.

```
lemma typeSafeDecl_43:
  assumes "tp = type.Storage (STMap t t')"
    and "Some (v, t'') = Some (KStoptr p, tv0)"
    and "cp = cp0"
    and "cd = cd0"
    and "state.Memory st = mem0"
    and "state.Storage st (Address ev) = sto0"
    and "(cd', mem', sck', ev') = (cdc', mem'', sck'', ev'')"
    and "Some (c, m', k', e) = (if tv0 = type.Storage (STMap t t') ∧ Denvalue ev'' $$ ip = None then
Some (cdc', mem'', astack_dup ip
        (type.Storage (STMap t t')) (KStoptr p) (sck'', ev'')) else None)"
  shows "declPost ev st cp mem' e c m' k'"
proof -
  let ?case = "declPost ev st cp mem' e c m' k'"
  note 43 = assms
  then have a10:"t'' = type.Storage (STMap t t')" using decl.simps(2) by (simp split:if_splits)
  have NoneIp:" Denvalue ev'' $$ ip = None" using 43 by (simp split:if_splits)
  then have a40:"Some (c, m', k', e) = Some (cdc', mem'', astack ip (type.Storage (STMap t t'))
(KStoptr p) (sck', ev''))"
    using 43 decl.simps(2) a10 typeSafeDecl_hyps(1) by simp
  then have k'Def:"k' = push (KStoptr p) sck'" by force

  have eDef:"e = updateEnv ip (type.Storage (STMap t t')) (Stackloc (ShowL_nat (Toploc sck'))
ev'')"using a40 unfolding astack.simps by blast
  then have eIP:"(Denvalue e) $$ ip = Some ((type.Storage (STMap t t')),(Stackloc (ShowL_nat (Toploc
sck'))))" by simp
  have sameADD:"Address ev' = Address e"
    using typeSafeDecl_hyps(4) decl_env by blast

  have wxDef:"tv0 = type.Storage (STMap t t')" using 43(8) by (simp split:if_splits)

  have a120:"lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
  then have a130:"∀ v'''. accessStore (ShowL_nat (Toploc sck')) sck' ≠ Some v'''"
    using lessThanTopLocs_def Read_Show_nat'_id using LSubPrefL2_def by auto
  then have a140:"∀ x y. ¬((Denvalue ev'') $$ x = Some y ∧ (snd y) = (Stackloc (ShowL_nat (Toploc
sck'))))"
    using TypeSafe_def typeSafeDecl_hyps(3) typeSafeAllStacklocsExist fmranI 43(7) by fastforce
  then have a150: "∀ ip''' t'''. Denvalue e $$ ip''' = Some (t''',(Stackloc (ShowL_nat (Toploc sck'))))
⟶ ip''' = ip"
    using a40 lessThanTopLocs_def fmranI by auto

  obtain locO tpO pO where oldAccess:"(
        (type.Storage tpO, locO) |∈| fmran (Denvalue ev') ∧
        (case locO of
         Stackloc stloc ⇒
           accessStore stloc sck' = Some (KStoptr pO) ∧ (tpO = (STMap t t') ∧ v = KStoptr pO ∨
extractValueType v ≠ pO
               ∧ CompStoType tpO (STMap t t') pO (extractValueType v))
         | Storeloc stloc ⇒ tpO = (STMap t t') ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧
CompStoType tpO (STMap t t') stloc (extractValueType v)))"
    using typeSafeDecl_hyps 43 wxDef by blast

  have compOld: " (∀ tp1 tp2 l1 l2 l1' l2' stl1 stl2.
        (type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
        (type.Storage tp2, l2) |∈| fmran (Denvalue ev') ∧
        (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
        (l2 = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2) ⟶
        (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if TypedStoSubpref
stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
    using typeSafeDecl_hyps(3) 43 unfolding TypeSafe_def compPointers_def by blast
  then have compOld':"(∀ tp1 l1 l1' l2' stl1 stl2.
```

```
        (type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
        (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
        (locO = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2) ∧ stl2 = pO ∨ locO = Storeloc
stl2) ⟶
        (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tpO stl1 stl2 else if TypedStoSubpref
stl1 stl2 tpO then CompStoType tpO tp1 stl2 stl1 else True))"
    using oldAccess by blast
  show ?thesis unfolding TypeSafe_def
  proof(intros)
    show "unique_locations (Denvalue e)" using typeSafeDecl_hyps(3) 43(7) updateEnvUniqueLocs a40 by
fastforce
  next
    have b2: "compPointers sck' (Denvalue ev')" using typeSafeDecl_hyps(3) 43(7) unfolding TypeSafe_def
by auto

    have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
    show "compPointers k' (Denvalue e)" unfolding compPointers_def
    proof intros
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume in1:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then obtain i1 i2 where iDef:" Denvalue e $$ i1 = Some (type.Storage tp1, l1) ∧ Denvalue e$$ i2
= Some (type.Storage tp2, l2)" by blast

      consider (bothStacks) "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1)
                  ∧ l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2)"
        | (StackLoc1) "l1 = Stackloc l1' ∧ accessStore l1' k' = Some (KStoptr stl1) ∧ l2 = Storeloc
stl2"
        | (StackLoc2) "l2 = Stackloc l2' ∧ accessStore l2' k' = Some (KStoptr stl2) ∧ l1 = Storeloc
stl1"
        | (StoreLocs) "l1 = Storeloc stl1 ∧ l2 = Storeloc stl2" using in1 by auto

      then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases)
        case bothStacks
        then show ?thesis
        proof(cases "l1 = Stackloc (ShowL_nat (Toploc sck'))")
          case True
          then have stl1IsP:"stl1 = p" using in1 k'Def typeSafeDecl_hyps 43 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
          have "i1 = ip" using iDef eDef a150 True by blast
          then have tp1IsStArray:"tp1 = STMap t t'" using iDef eDef by auto
          then show ?thesis
          proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
            case True
            then have stl2IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 43 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
            have "i2 = ip" using iDef eDef a150 True by blast
            then have tp2IsStArray:"tp2 = STMap t t'" using iDef eDef by auto
            then show ?thesis using stl2IsP tp2IsStArray tp1IsStArray stl1IsP by auto
          next
            case False
            then have acc2:"accessStore l2' k' = accessStore l2' sck'"
              using k'Def bothStacks
              by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)
            then have i2NotIP:"i2 ≠ ip" using eDef iDef False by auto
            then have tp2InOld:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
              using fmranI iDef by fastforce
            have sameAcc2:" accessStore l2' sck' = Some (KStoptr stl2)" using acc2 in1 bothStacks by
auto
```

```
            then show ?thesis
            proof(cases loc0)
              case (Stackloc x1)
              then have acc:"accessStore x1 sck' = Some (KStoptr p0)" using oldAccess by simp
              then have cc:"(tp0 = STMap t t' ∧ v = KStoptr p0 ∨ extractValueType v ≠ p0 ∧
CompStoType tp0 (STMap t t') p0 (extractValueType v))"
                using compOld oldAccess using 43(7) Stackloc by simp
              have comps:"(if TypedStoSubpref p0 stl2 tp2 then CompStoType tp2 tp0 stl2 p0 else
                           if TypedStoSubpref stl2 p0 tp0 then CompStoType tp0 tp2 p0 stl2 else True)"
                using compOld' sameAcc2 bothStacks tp2InOld Stackloc 43(7) acc by blast
              then show ?thesis
              proof(cases "v = KStoptr p0")
                case True

                then have "tp0 = STMap t t'" using cc True by simp
                then have sames:"stl1 = p0 ∧ tp1 = tp0" using tp1IsStArray True 43 stl1IsP in1 by
blast

                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tp0 (STMap t t') p0 stl1" using stl1IsP cc 43 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
              qed
            next
              case (Storeloc x2)
              then have cc:"(tp0 = STMap t t' ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧
CompStoType tp0 (STMap t t') x2 (extractValueType v))"
                using compOld oldAccess using 43(7) by simp
              have comps:"(if TypedStoSubpref x2 stl2 tp2 then CompStoType tp2 tp0 stl2 x2 else
                           if TypedStoSubpref stl2 x2 tp0 then CompStoType tp0 tp2 x2 stl2 else True)"
                using compOld' sameAcc2 bothStacks tp2InOld Storeloc 43(7) by blast
              then show ?thesis
              proof(cases "v = KStoptr x2")
                case True

                then have "tp0 = STMap t t'" using cc True by simp
                then have sames:"stl1 = x2∧ tp1 = tp0" using tp1IsStArray True 43 stl1IsP in1 by
blast

                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tp0 (STMap t t') x2 stl1" using stl1IsP cc 43 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
              qed

            qed
          qed
        next
          case False
          then have acc1:"accessStore l1' k' = accessStore l1' sck'"
            using k'Def bothStacks
            by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)

          then have i1NotIp:"i1 ≠ ip" using False eDef iDef by auto
          then have In''1:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
```

```
            using fmranI iDef by fastforce
          then show ?thesis
          proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
            case True
            then have stl2IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 43 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
            have "i2 = ip" using iDef eDef a150 True by blast
            then have tp2IsStArray:"tp2 = STMap t t'" using iDef eDef by auto
            have tp1InOld:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev')" using In''1 43(7) by blast
            then show ?thesis
            proof(cases locO)
              case (Stackloc x1)
              then have acc:"accessStore x1 sck' = Some (KStoptr pO)" using oldAccess by simp
              then have cc:"(tpO = STMap t t' ∧ v = KStoptr pO ∨ extractValueType v ≠ pO ∧
CompStoType tpO (STMap t t') pO (extractValueType v))"
                using compOld oldAccess using 43(7) Stackloc by simp
              have comps:"(if TypedStoSubpref pO stl1 tp1 then CompStoType tp1 tpO stl1 pO else
                           if TypedStoSubpref stl1 pO tpO then CompStoType tpO tp1 pO stl1 else True)"
                using compOld' acc1 in1 bothStacks tp1InOld Stackloc 43(7) acc by metis
              then show ?thesis
              proof(cases "v = KStoptr pO")
                case True

                then have "tpO = STMap t t'" using cc True by simp
                then have sames:"stl2 = pO ∧ tp2 = tpO" using tp2IsStArray True 43 stl2IsP in1 by
blast

                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tpO (STMap t t') pO stl2" using stl2IsP cc 43 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp2IsStArray by (simp
only:)
              qed
            next
              case (Storeloc x2)
              then have cc:"(tpO = STMap t t' ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧
CompStoType tpO (STMap t t') x2 (extractValueType v))"
                using compOld oldAccess using 43(7) by simp
              have comps:"(if TypedStoSubpref x2 stl1 tp1 then CompStoType tp1 tpO stl1 x2 else
                           if TypedStoSubpref stl1 x2 tpO then CompStoType tpO tp1 x2 stl1 else True)"
                using compOld' acc1 in1 bothStacks tp1InOld Storeloc 43(7) by metis
              then show ?thesis
              proof(cases "v = KStoptr x2")
                case True
                then have "tpO = STMap t t'" using cc True by simp
                then have sames:"stl2 = x2∧ tp2 = tpO" using tp2IsStArray True 43 stl2IsP in1 by
blast

                then show ?thesis using compPointers_if_sym[OF comps] comps by simp
              next
                case False
                then have comps2:"CompStoType tpO (STMap t t') x2 stl2" using stl2IsP cc 43 by simp

                then show ?thesis using compPointers_if_child[OF comps comps2]
                  compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp2IsStArray by (simp
only:)
              qed
            qed
          next
            case f2:False
            then have acc2:"accessStore l2' k' = accessStore l2' sck'"
```

```
            using k'Def bothStacks
            by (metis accessStore_def accessStore_non_changed allocateMapping k'Def push_def snd_conv
surj_pair)
          then have i2NotIP:"i2 ≠ ip" using eDef iDef f2 by auto
          then have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
            using fmranI iDef by fastforce
          have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev') ∧
    (type.Storage tp2, l2) |∈| fmran (Denvalue ev') ∧
    (l1 = Stackloc l1' ∧ accessStore l1' sck' = Some (KStoptr stl1)) ∧
    (l2 = Stackloc l2' ∧ accessStore l2' sck' = Some (KStoptr stl2))" using 43(7) In''2 In''1
bothStacks acc1 acc2 in1 by auto
          then show ?thesis using b2 unfolding compPointers_def by blast
        qed
      qed
     next
      case StackLoc1
      then have "i2 ≠ ip" using iDef eDef by auto
      then have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef
        using fmranI iDef by fastforce
      then show ?thesis
      proof(cases "l1 = Stackloc (ShowL_nat (Toploc sck'))")
        case True
        then have stl1IsP:"stl1 = p" using in1 k'Def typeSafeDecl_hyps 43 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
        have "i1 = ip" using iDef eDef a150 True by blast
        then have tp1IsStArray:"tp1 = STMap t t'" using iDef eDef by auto
        then show ?thesis
        proof(cases loc0)
          case (Stackloc x1)
          then have acc:"accessStore x1 sck' = Some (KStoptr p0)" using oldAccess by simp
          then have cc:"(tp0 = STMap t t' ∧ v = KStoptr p0 ∨ extractValueType v ≠ p0 ∧ CompStoType
tp0 (STMap t t') p0 (extractValueType v))"
            using comp0ld oldAccess using 43(7) Stackloc by simp
          have comps:"(if TypedStoSubpref p0 stl2 tp2 then CompStoType tp2 tp0 stl2 p0 else
                        if TypedStoSubpref stl2 p0 tp0 then CompStoType tp0 tp2 p0 stl2 else True)"
            using comp0ld' In''2 Stackloc 43(7) acc StackLoc1 by blast
          then show ?thesis
          proof(cases "v = KStoptr p0")
            case True

            then have "tp0 = STMap t t'" using cc True by simp
            then have sames:"stl1 = p0 ∧ tp1 = tp0" using tp1IsStArray True 43 stl1IsP in1 by blast

            then show ?thesis using compPointers_if_sym[OF comps] comps by simp
          next
            case False
            then have comps2:"CompStoType tp0 (STMap t t') p0 stl1" using stl1IsP cc 43 by simp
            then show ?thesis using compPointers_if_child[OF comps comps2]
              compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
          qed
        next
          case (Storeloc x2)
          then have cc:"(tp0 = STMap t t' ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧ CompStoType
tp0 (STMap t t') x2 (extractValueType v))"
            using comp0ld oldAccess using 43(7) by simp
          have comps:"(if TypedStoSubpref x2 stl2 tp2 then CompStoType tp2 tp0 stl2 x2 else
                        if TypedStoSubpref stl2 x2 tp0 then CompStoType tp0 tp2 x2 stl2 else True)"
            using comp0ld' StackLoc1 In''2 Storeloc 43(7) by blast
          then show ?thesis
          proof(cases "v = KStoptr x2")
            case True
            then have "tp0 = STMap t t'" using cc True by simp
            then have sames:"stl1 = x2∧ tp1 = tp0" using tp1IsStArray True 43 stl1IsP in1 by blast
```

```
            then show ?thesis using compPointers_if_sym[OF comps] comps by simp
          next
            case False
            then have comps2:"CompStoType tp0 (STMap t t') x2 stl1" using stl1IsP cc 43 by simp
            then show ?thesis using compPointers_if_child[OF comps comps2]
              compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
          qed
        qed
      next
        case False
        then have "i1 ≠ ip" using iDef eDef by auto
        then have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')"
          using eDef iDef fmranI by fastforce
        moreover have "accessStore l1' sck' = Some (KStoptr stl1)"
          using StackLoc1 k'Def False unfolding push_def allocate_def accessStore_def updateStore_def
by auto
        ultimately show ?thesis using compOld 43(7) StackLoc1 using In''2 by blast
      qed
    next
      case StackLoc2
      then have "i1 ≠ ip" using iDef eDef by auto
      then have In''2:"(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
        using fmranI iDef by fastforce
      then show ?thesis
      proof(cases "l2 = Stackloc (ShowL_nat (Toploc sck'))")
        case True
        then have stl1IsP:"stl2 = p" using in1 k'Def typeSafeDecl_hyps 43 unfolding push_def
allocate_def accessStore_def updateStore_def by auto
        have "i2 = ip" using iDef eDef a150 True by blast
        then have tp1IsStArray:"tp2 = STMap t t'" using iDef eDef by auto
        then show ?thesis
        proof(cases loc0)
          case (Stackloc x1)
          then have acc:"accessStore x1 sck' = Some (KStoptr p0)" using oldAccess by simp
          then have cc:"(tp0 = STMap t t' ∧ v = KStoptr p0 ∨ extractValueType v ≠ p0 ∧ CompStoType
tp0 (STMap t t') p0 (extractValueType v))"
            using compOld oldAccess using 43(7) Stackloc by simp
          have comps:"(if TypedStoSubpref p0 stl1 tp1 then CompStoType tp1 tp0 stl1 p0 else
                        if TypedStoSubpref stl1 p0 tp0 then CompStoType tp0 tp1 p0 stl1 else True)"
            using compOld' In''2 Stackloc 43(7) acc StackLoc2 by blast
          then show ?thesis
          proof(cases "v = KStoptr p0")
            case True

            then have "tp0 = STMap t t'" using cc True by simp
            then have sames:"stl2 = p0 ∧ tp2 = tp0" using tp1IsStArray True 43 stl1IsP in1 by blast

            then show ?thesis using compPointers_if_sym[OF comps] comps by simp
          next
            case False
            then have comps2:"CompStoType tp0 (STMap t t') p0 stl2" using stl1IsP cc 43 by simp

            then show ?thesis using compPointers_if_child[OF comps comps2]
              compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
          qed
        next
          case (Storeloc x2)
          then have cc:"(tp0 = STMap t t' ∧ v = KStoptr x2 ∨ extractValueType v ≠ x2 ∧ CompStoType
tp0 (STMap t t') x2 (extractValueType v))"
            using compOld oldAccess using 43(7) by simp
          have comps:"(if TypedStoSubpref x2 stl1 tp1 then CompStoType tp1 tp0 stl1 x2 else
```

```
                        if TypedStoSubpref stl1 x2 tp0 then CompStoType tp0 tp1 x2 stl1 else True)"
          using compOld' StackLoc2 In''2 Storeloc 43(7) by blast
        then show ?thesis
        proof(cases "v = KStoptr x2")
          case True
          then have "tp0 = STMap t t'" using cc True by simp
          then have sames:"stl2 = x2∧ tp2 = tp0" using tp1IsStArray True 43 stl1IsP in1 by simp

          then show ?thesis using compPointers_if_sym[OF comps] comps by simp

        next
          case False
          then have comps2:"CompStoType tp0 (STMap t t') x2 stl2" using stl1IsP cc 43 by simp
          then show ?thesis using compPointers_if_child[OF comps comps2]
            compPointers_if_sym[OF compPointers_if_child[OF comps comps2]] tp1IsStArray by (simp
only:)
        qed
      qed
    next
      case False
      then have "i2 ≠ ip" using iDef eDef by auto
      then have "(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')"
        using eDef iDef fmranI by fastforce
      moreover have "accessStore l2' sck' = Some (KStoptr stl2)"
        using StackLoc2 k'Def False unfolding push_def allocate_def accessStore_def updateStore_def
by auto
      ultimately show ?thesis using compOld 43(7) StackLoc2 using In''2 by blast
    qed
  next
    case StoreLocs
    then have "i1 ≠ ip" using iDef eDef by auto
    moreover have "(type.Storage tp1, l1) |∈| fmran (Denvalue ev'')" using eDef
      using fmranI iDef calculation by fastforce
    moreover have "i2 ≠ ip" using iDef eDef StoreLocs by auto
    moreover have In''2:"(type.Storage tp2, l2) |∈| fmran (Denvalue ev'')" using eDef calculation
      using fmranI iDef by fastforce
    ultimately show ?thesis using compOld StoreLocs 43(7) by blast
  qed
qed
next
  have *:"safeContract (Accounts st) (Storage st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def
using 43(7) by auto
  have **:"Address ev' = Address e" using eDef 43(7) by simp
  have ***:"Contract ev' = Contract e" using eDef 43(7) by simp
  show "safeContract (Accounts st) (Storage st) " using * by auto
next
  show "balanceTypes (Accounts st)" using typeSafeDecl_hyps(1) TypeSafe_def by simp
next
  show "svalueTypes (Svalue e)" using eDef 43(7) typeSafeDecl_hyps(3) TypeSafe_def svalueTypes_def
by simp
next
  have "lessThanTopLocs sck'" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
  then show "lessThanTopLocs k'" using stackPushToplocSafe k'Def 43(7) by metis
next
  have "lessThanTopLocs cdc'" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
  then show "lessThanTopLocs c" using stackPushToplocSafe 43(7) a40 by simp
next
  have "lessThanTopLocs mem''" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
  then show "lessThanTopLocs m'" using 43(7) a40 by simp
next
  have "addressFormat (Address ev')" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
  then show "addressFormat(Address e)" using 43(7) a40 by auto
next
  have "addressFormat (Sender ev')" using typeSafeDecl_hyps(3) TypeSafe_def 43(7) by simp
```

```
    then show "addressFormat (Sender e)" using 43(7) a40 by auto
  next
    show "typeCompat (Denvalue e) k' m' (Storage st (Address e)) c " unfolding typeCompat_def
    proof intros
      fix tDen lDen
      assume *: "(tDen, lDen) |∈| fmran (Denvalue e)"
      then obtain ip'' where a90:"Denvalue e $$ ip'' = Some (tDen, lDen)" using * by auto
      then have a100:"(Storage st (Address ev')) = (Storage st (Address e))" using
typeSafeDecl_hyps(3) eDef
        using sameADD by presburger
      have a110:"m' = mem''" and a115:"cdc' = c" using a40 by simp+

      then have a160:"∀ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using eDef lessThanTopLocs_def fmranI by auto
      show "denvalueCon k' m' (Storage st (Address e)) c tDen lDen"
      proof(cases lDen)
        case (Stackloc x1)
        then show ?thesis
        proof (cases "x1 = ShowL_nat (Toploc sck')")
          case True
          then have a170: "accessStore (ShowL_nat (Toploc sck')) k' = Some (KStoptr p)"
            using k'Def by (simp add:push_def allocate_def updateStore_def accessStore_def)

          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using True a170 Stackloc by simp
          next
            case some:(Some a)
            then have a180:"a = (KStoptr p) " using a170 True by simp
            then have "ip'' = ip" using a150 True a90 Stackloc by simp
            then have "tDen = type.Storage (STMap t t')" using a90 eDef by simp
            moreover have "(∀locs tp. SCon tp locs (Storage st (Address ev)) ⟶ SCon tp locs (Storage
st (Address ev')))"
              using typeSafeDecl_hyps(8) 43 wxDef by simp
            moreover have "SCon (STMap t t') p (Storage st (Address e))"
              using typeSafeDecl_hyps(2) 43(2) wxDef calculation sameADD by simp
            ultimately show ?thesis using Stackloc some a180 by simp
          qed
        next
          case False
          then have "ip'' ≠ ip" using a160 a90 Stackloc by simp
          then have a170: "(tDen, lDen) |∈| fmran (Denvalue ev')" using eDef Stackloc fmranI a90 False
* 43(7) by fastforce
          then obtain y' where a180:"accessStore x1 sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) Stackloc 43(7) by blast
          then have a190:"accessStore x1 k' = accessStore x1 sck'" using k'Def False by (simp
add:push_def allocate_def updateStore_def accessStore_def)
          then show ?thesis
          proof(cases "accessStore x1 k'")
            case None
            then show ?thesis using a180 a190 Stackloc by simp
          next
            case some:(Some a)
            then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def
              using a190 some Stackloc a170 * a180 a110 a115 a100 43(7) by (cases a; cases tDen;
force+)
          qed
        qed
      next
        case (Storeloc x2)
        then have a170:"ip'' ≠ ip" using eDef Storeloc a90 by auto
        then have a180: "(tDen, lDen) |∈| fmran (Denvalue ev')" using eDef 43(7) Storeloc fmranI a90
```

```
by fastforce
        then show ?thesis using typeSafeDecl_hyps(3) unfolding TypeSafe_def typeCompat_def using
Storeloc a100 43(7) by (cases tDen;force)
      qed
    qed
  next
    show "AddressTypes (Accounts st)" using typeSafeDecl_hyps(3) unfolding TypeSafe_def by simp
  next

    have a110:"m' = mem''" using a40 by simp
    then show "⋀locs tp. ¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m' locs" using
typeSafeDecl_hyps(5) 43(7) typeSafeDecl_hyps(5) by (simp)
  next
    assume "¬ cp"
    then show "Toploc (Memory st) ≤ Toploc m'" using typeSafeDecl_hyps(5) 43(7) a40 by blast
  next
    assume notCP:"¬cp"
    show "ncpDenvalueLimit e ev k' (Stack st) (Memory st) "
      unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p'' i
      assume a120:" (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e) ∧ accessStore locs k' =
Some (KMemptr
          p'') ∧ i < Toploc (Memory st) ∧ LSubPrefL2 p'' (ShowL_nat i)"

      then have a160:"∀ip''' t''' l'''. Denvalue e $$ ip''' = Some(t''', l''') ∧
          l''' ≠ (Stackloc (ShowL_nat (Toploc sck'))) ⟶ ip''' ≠ ip"
        using eDef lessThanTopLocs_def fmranI by auto
      show " ∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p'' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'' ∧
              tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p''))
          "
      proof -
        obtain ip'' where ip''def:"(Denvalue e) $$ ip'' = Some (type.Memory tp', Stackloc locs)" using
a120 by blast
        then have a170:"ip'' ≠ ip" using eDef by auto
        then have a180: "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue ev')" using eDef 43(7)
fmranI ip''def by fastforce
        have a190:"locs ≠ (ShowL_nat (Toploc sck'))" using a160 a170 k'Def
          using a150 ip''def by blast

        then obtain y' where a200:"accessStore locs sck' = Some y' " using typeSafeAllStacklocsExist
typeSafeDecl_hyps(3) 43(7) a180 by blast
        then have "accessStore locs k' = accessStore locs sck'" using k'Def a120 a190 unfolding
push_def allocate_def updateStore_def accessStore_def by simp
        then show " ∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue ev) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p'' ∧ tp'' = tp' ∨ (∃len arr. p' ≠
p'' ∧
              tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr tp' p' p''))"
          using typeSafeDecl_hyps(5) 43(7) a120 a40 eDef k'Def a120 a180 notCP ncpDenvalueLimit_def
          by (metis)
      qed
    qed
  next
    assume notCP:"¬cp"
    then show "ncpOMemInDMem (Memory st) m'" using a40 43(7) typeSafeDecl_hyps by simp
  next
    assume notCP:"¬cp"
    then show "ncpElementsNoSubPref (Memory st) m'" using a40 typeSafeDecl_hyps 43 by blast
  next
    assume notCP:"¬cp"
```

```
    then show "ncpNewSelfPoint (Memory st) m'" using a40 typeSafeDecl_hyps 43 by blast
  next
    show "Toploc mem' ≤ Toploc m' " using a40 43 by simp
  next
    show "denvalueTypeCorrectness e k' m'" unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t l ptr_loc
      assume *:" (type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some (KMemptr
ptr_loc)"
      then have "(type.Memory t, Stackloc l) |∈| fmran (Denvalue ev')"
        using a40 unfolding astack.simps updateEnv.simps using type.distinct(7) typeSafeDecl_hyps(4)
decl_env_not_i fmlookup_ran_iff
          option.inject prod.inject
        by fastforce
      moreover have "accessStore l k' = accessStore l sck'"
        using a40 * unfolding astack.simps push_def updateStore_def accessStore_def allocate_def
        by (metis (no_types, lifting) accessStore_def typeSafeDecl_hyps(3,4) calculation
decl_stack_change
            fmlookup_ran_iff snd_eqD TypeSafe_def stackLocs_imp_NotDen)
      ultimately show "case t of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) m' = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc m' = Some (MTValue val)"
        using a40 typeSafeDecl_hyps(3) * unfolding TypeSafe_def denvalueTypeCorrectness_def using 43(7)
by auto
    qed
  next
    show "subPrefixStructuralConsistency m'"
      using typeSafeDecl_hyps(3) a40 43(7) unfolding TypeSafe_def subPrefixStructuralConsistency_def by
auto
  next
    show "SomeValSomeTyp m'" using typeSafeDecl_hyps(3) unfolding TypeSafe_def using a40 43(7) by
blast
  qed
qed


lemma typeSafeDecl:
  shows "declPost ev st cp mem' e c m' k'" using typeSafeDecl_hyps(4)
proof(cases rule:decl.elims)
  case (1 t _ _ _ _ c m k e)
  then show ?thesis using typeSafeDecl_hyps by blast
next
  case (2 t v t' _ _ _ _ cd' mem'' sck'' ev'')
  from 2 show ?thesis by (rule typeSafeDecl_2)
next
  case 3
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 4
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 5
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 6
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 7
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 8
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case (9 x t p _ cd _ _ cd' mem'' sck'' ev'')
```

```
  from 9 show ?thesis by (rule typeSafeDecl_9)
next
  case (10 x t p _ _ mem _ cd' mem''' sck'' ev'')
  from 10 show ?thesis by (rule typeSafeDecl_10)
next
  case (11 v _ _ _ _ _ _)
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 12
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 13
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case (14 v _ _ _ _ _ _ _)
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 15
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 16
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case 17
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case (18 x t _ _ _ _ c m k e)
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case (19 x t p _ _ mem _ cd' mem''' sck'' ev'')
  from 19 show ?thesis by (rule typeSafeDecl_19)
next
  case (20 x t p _ _ _ _ cd' mem''' sck'' ev'')
  from 20 show ?thesis by (rule typeSafeDecl_20)
next
  case (21 x t p _ _ cd _ _ cd' mem''' sck'' ev'')
  from 21 show ?thesis by (rule typeSafeDecl_21)
next
  case (22 x t p x' t' _ _ _ s cd' mem''' sck'' ev'')
  from 22 show ?thesis by (rule typeSafeDecl_22)
next
  case (23 v _ _ _ _ _ _)
  then show ?thesis using typeSafeDecl_hyps(1) by simp
next
  case (24 _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case 25
  then show ?thesis by simp
next
  case (26 v _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (27 v _ _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (28 _ _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case (29 _ _ v _ _ _ _ _)
  then show ?thesis by simp
next
  case (30 _ _ v _ _ _ _ _)
  then show ?thesis by simp
```

```
next
  case 31
  then show ?thesis by simp
next
  case 32
  then show ?thesis by simp
next
  case (33 _ v _ _ _ _)
  then show ?thesis by simp
next
  case 34
  then show ?thesis by simp
next
  case (35 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (36 v _ _ _ _ _ _)
  then show ?thesis by simp
next
  case (37 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case (38 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case (39 _ _ v _ _ _ _)
  then show ?thesis by simp
next
  case 40
  then show ?thesis by simp
next
  case 41
  then show ?thesis by simp
next
  case (42 x t p _ _ _ _ _ _ cd' mem'' sck'' ev'')
  from 42 show ?thesis by (rule typeSafeDecl_42)
next
  case (43 t t' p _ _ _ _ _ _ cd' mem'' sck'' ev'')
  from 43 show ?thesis by (rule typeSafeDecl_43)
next
  case (44 v _ xk xl xm xn xo)
  then show ?thesis by simp
next
  case (45 v _ _ _ xk xl xm xn xo)
  then show ?thesis by simp
next
  case (46 v _ _ _ xk xl xm xn xo)
  then show ?thesis by simp
next
  case (47 v _ _ _ xk xl xm xn xo)
  then show ?thesis by simp
next
  case (48 v xj xk xl xm xn xo)
  then show ?thesis by simp
next
  case (49 xi xk xl xm xn xo)
  then show ?thesis by simp
next
  case (50 xi _ _ xk xl xm xn xo)
  then show ?thesis by simp
next
  case (51 xi _ _ xk xl xm xn xo)
  then show ?thesis by simp
next
```

```
  case (52 xi _ _ xk xl xm xn xo)
  then show ?thesis by simp
qed


end

end

end
```

14 Expression preservation (TypeSafe_Expressions)

```
theory TypeSafe_Expressions
  imports TypeSafe_Copies TypeSafe_Decl
begin
```

The supported Solidity expressions do not violate the type consistency of a given environment.

```
context typesafe_base
begin
```

14.1 Olift and Plift

```
lemma oliftTypeInput:
  assumes "olift op t1 t2 v1 v2 = Some(v',t')"
  shows "(∃ b. t1 = (TSInt b) ∨ t1 = (TUInt b)) ∧ (∃ b2. t2 = (TSInt b2) ∨ t2 = (TUInt b2))"
proof
  show "∃ b. t1 = TSInt b ∨ t1 = TUInt b" using assms olift.simps by (cases t1; cases t2; simp
split:if_split_asm)
next
  show "∃ b2. t2 = TSInt b2 ∨ t2 = TUInt b2" using assms olift.simps by (cases t1; cases t2; simp
split:if_split_asm)
qed

lemma pliftTypeInput:
  assumes "plift op t1 t2 v1 v2 = Some(v',t')"
  shows "(∃ b. t1 = (TSInt b) ∨ t1 = (TUInt b)) ∧ (∃ b2. t2 = (TSInt b2) ∨ t2 = (TUInt b2))"
proof
  show "∃ b. t1 = TSInt b ∨ t1 = TUInt b" using assms plift.simps by (cases t1; cases t2; simp
split:if_split_asm)
next
  show "∃ b2. t2 = TSInt b2 ∨ t2 = TUInt b2" using assms plift.simps by (cases t1; cases t2; simp
split:if_split_asm)
qed

lemma vtandTypeOut:
  assumes "vtand t1 t2 v1 v2 = Some(v', t')"
  shows "(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' = TBool"
proof -
  have a10:"t1 = TBool ∧ t2 = TBool" using assms vtand.elims by blast
  have "vtand TBool TBool v1 v2 = Some (ShowL_bool True, TBool) ∨ vtand TBool TBool v1 v2 = Some
      (ShowL_bool False, TBool)" using vtand.simps assms by simp
  then show "(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' = TBool" using assms a10 by auto
qed

lemma vtorTypeOut:
  assumes "vtor t1 t2 v1 v2 = Some(v', t')"
  shows "(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' = TBool"
proof -
  have a10:"t1 = TBool ∧ t2 = TBool" using assms vtor.elims by blast
  have "vtor TBool TBool v1 v2 = Some (ShowL_bool True, TBool) ∨ vtor TBool TBool v1 v2 = Some
      (ShowL_bool False, TBool)" using vtor.simps assms by simp
  then show "(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' = TBool" using assms a10 by auto
```

qed

lemma *vtandTypeCon:*
assumes *"vtand t1 t2 v1 v2 = Some(v', t')"*
and *"(v' = ShowL$_{bool}$ True ∨ v' = ShowL$_{bool}$ False) ∧ t' = TBool"*
shows *"typeCon t' v'"* **using** *checkBool_def createBool_def ShowL$_{bool}$_def assms* **by** *(simp split:if_splits)*

14.1.1 Results of successful olift must be typecon

lemma *oliftTypeCon:*
assumes *"olift op t1 t2 v1 v2 = Some(v',t')"*
and *"t1 = TSInt b1 ∨ t1 = TUInt b1"*
and *"t2 = TSInt b2 ∨ t2 = TUInt b2"*
shows *"typeCon t' v'"*
proof -
consider *(s1) "t1 = TSInt b1" | (u1) "t1 = TUInt b1"* **using** *assms(2)* **by** *auto*
then show *?thesis*
proof*(cases)*
case *s1*
consider *(s2) "t2 = TSInt b2" | (u2) "t2 = TUInt b2"* **using** *assms(3)* **by** *auto*
then show *?thesis*
proof*(cases)*
case *s2*
then have *a10:"v' = createSInt (max b1 b2) (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"* **using** *s1 olift.simps(1) assms* **by** *simp*
then have *a20:"t' = TSInt (max b1 b2)"* **using** *s1 s2 olift.simps(1) assms* **by** *simp*
then have *"checkSInt (max b1 b2) (createSInt (max b1 b2) (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2)))"*
using *checkSIntCreate[of "(max b1 b2)" "(op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"] assms(3)* **by** *simp*
then show *?thesis* **using** *a10 a20* **by** *simp*
next
case *u2*
then have *a10:"v' = createSInt b1 (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"* **using** *s1 olift.simps(3) assms* **by** *(simp split:if_split_asm)*
then have *a20:"t' = TSInt b1"* **using** *s1 u2 olift.simps(3) assms* **by** *(simp split:if_split_asm)*
then have *"checkSInt b1 (createSInt b1 (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2)))"*
using *checkSIntCreate[of "b1" "(op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"] assms(3)* **by** *simp*
then show *?thesis* **using** *a10 a20* **by** *simp*
qed
next
case *u1*
consider *(s2) "t2 = TSInt b2" | (u2) "t2 = TUInt b2"* **using** *assms(3)* **by** *auto*
then show *?thesis*
proof*(cases)*
case *s2*
then have *a10:"v' = createSInt b2 (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"* **using** *u1 olift.simps(4) assms* **by** *(simp split:if_split_asm)*
then have *a20:"t' = TSInt b2"* **using** *u1 s2 olift.simps(4) assms* **by** *(simp split:if_split_asm)*
then have *"checkSInt b2 (createSInt b2 (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2)))"*
using *checkSIntCreate[of "b2" "(op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"] assms(3)* **by** *simp*
then show *?thesis* **using** *a10 a20* **by** *simp*
next
case *u2*
then have *a10:"v' = createUInt (max b1 b2) (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"* **using** *u1 olift.simps(2) assms* **by** *simp*
then have *a20:"t' = TUInt (max b1 b2)"* **using** *u1 u2 olift.simps(2) assms* **by** *simp*
then have *"checkUInt (max b1 b2) (createUInt (max b1 b2) (op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2)))"*
using *checkUIntCreate[of "(max b1 b2)" "(op (ReadL$_{int}$ v1) (ReadL$_{int}$ v2))"] assms(3)* **by** *simp*
then show *?thesis* **using** *a10 a20* **by** *simp*
qed
qed
qed

14.1.2 Results of successful plift must be typecon

```
lemma pliftTypeCon:
  assumes "plift op t1 t2 v1 v2 = Some(v',t')"
    and "t1 = TSInt b1 ∨ t1 = TUInt b1"
    and "t2 = TSInt b2 ∨ t2 = TUInt b2"
  shows "typeCon t' v'"
proof -
  consider (s1) "t1 = TSInt b1" | (u1) "t1 = TUInt b1" using assms(2) by auto
  then show ?thesis
  proof(cases)
    case s1
    consider (s2) "t2 = TSInt b2" | (u2) "t2 = TUInt b2" using assms(3) by auto
    then show ?thesis
    proof(cases)
      case s2
      then have a10:"v' = createBool (op (ReadL_int v1) (ReadL_int v2))" using s1 plift.simps(1) assms
by simp
      then have a20:"t' = TBool" using s1 s2 plift.simps(1) assms by simp
      then have "checkBool (createBool (op (ReadL_int v1) (ReadL_int v2)))" unfolding checkBool_def
          createBool_def using ShowL_bool_def by simp
      then show ?thesis using a10 a20 by simp
    next
      case u2
      then have a10:"v' = createBool (op (ReadL_int v1) (ReadL_int v2))" using s1 plift.simps(3) assms
by (simp split:if_split_asm)
      then have a20:"t' = TBool" using s1 u2 plift.simps(3) assms by (simp split:if_split_asm)
      then have "checkBool (createBool (op (ReadL_int v1) (ReadL_int v2)))" unfolding checkBool_def
          createBool_def using ShowL_bool_def by simp
      then show ?thesis using a10 a20 by simp
    qed
  next
    case u1
    consider (s2) "t2 = TSInt b2" | (u2) "t2 = TUInt b2" using assms(3) by auto
    then show ?thesis
    proof(cases)
      case s2
      then have a10:"v' = createBool (op (ReadL_int v1) (ReadL_int v2))" using u1 plift.simps(4) assms
by (simp split:if_split_asm)
      then have a20:"t' = TBool" using u1 s2 plift.simps(4) assms by (simp split:if_split_asm)
      then have "checkBool (createBool (op (ReadL_int v1) (ReadL_int v2)))" unfolding checkBool_def
          createBool_def using ShowL_bool_def by simp
      then show ?thesis using a10 a20 by simp
    next
      case u2
      then have a10:"v' = createBool (op (ReadL_int v1) (ReadL_int v2))" using u1 plift.simps(2) assms
by simp
      then have a20:"t' = TBool" using u1 u2 plift.simps(2) assms by simp
      then have "checkBool (createBool (op (ReadL_int v1) (ReadL_int v2)))" unfolding checkBool_def
          createBool_def using ShowL_bool_def by simp
      then show ?thesis using a10 a20 by simp
    qed
  qed
qed
```

14.1.3 Lifting an expression with olift yields typecon

```
lemma oliftSuccess:
  assumes "lift expr (olift op) e1 e2 ev cd st g = Normal ((KValue v, Value t), g')"
  shows "typeCon t v"
proof -
  obtain v1 t1 g'' where a10:"expr e1 ev cd st g = Normal((KValue v1, Value t1), g'')"
    using lift_def assms by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
  then obtain v2 t2 g''' where a20:"expr e2 ev cd st g'' = Normal((KValue v2, Value t2), g''')" using
lift_def assms by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
```

```
  then have a30:"(olift op) t1 t2 v1 v2 = Some(v, t)" using a10 a20 assms lift_def by (auto
split:result.split_asm stackvalue.split_asm type.split_asm option.split_asm)
  then obtain b1 where a40:"t1 = TSInt b1 ∨ t1 = TUInt b1" using oliftTypeInput[of op t1 t2 v1 v2 v
t] a10 by auto
  then obtain b2 where a50:"t2 = TSInt b2 ∨ t2 = TUInt b2" using oliftTypeInput[of op t1 t2 v1 v2 v
t] a20 a30 by auto
  then show ?thesis using oliftTypeCon[of op t1 t2 v1 v2 v t b1 b2] a30 a40 a50 by simp
qed
```

14.1.4 Lifting an expression with plift yields typecon

```
lemma pliftSuccess:
  assumes "lift expr (plift op) e1 e2 ev cd st g = Normal ((KValue v, Value t), g')"
  shows "typeCon t v"
proof -
  obtain v1 t1 g'' where a10:"expr e1 ev cd st g = Normal((KValue v1, Value t1), g'')" using lift_def
assms by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
  then obtain v2 t2 g''' where a20:"expr e2 ev cd st g'' = Normal((KValue v2, Value t2), g''')" using
lift_def assms by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
  then have a30:"(plift op) t1 t2 v1 v2 = Some(v, t)" using a10 a20 assms lift_def by (auto
split:result.split_asm stackvalue.split_asm type.split_asm option.split_asm)
  then obtain b1 where a40:"t1 = TSInt b1 ∨ t1 = TUInt b1" using pliftTypeInput[of op t1 t2 v1 v2 v
t] a10 by auto
  then obtain b2 where a50:"t2 = TSInt b2 ∨ t2 = TUInt b2" using pliftTypeInput[of op t1 t2 v1 v2 v
t] a20 a30 by auto
  then show ?thesis using pliftTypeCon[of op t1 t2 v1 v2 v t b1 b2] a30 a40 a50 by simp
qed
```

14.2 Proof that expressions do not violate type consistency

The following lemma proves that given a typesafe environment all expressions, Memory selections, Storage selections and rexpressions return type consistent values, i.e. the returned string values conform to the expected types. This demonstrates that the semantics of expressions do not alter the values in any way which may violate type consistency. Further, the environment, memory, calldata, and storage created by `local.load` are type-safe with respect to one another.

The environment is assumed to be fully initialised by `ffold_init`: the contract exists in the accounts and all contract variables are initialised. This holds in the context of statements.

```
lemma exprTypeconInduct_aux:
  "∀l1' t1' g1' arr. msel c1 t1 l1 xe1 ev1 cd1 st1 g1 = Normal ((l1', t1'), g1') ∧
                           MCon t1 (if c1 then Memory st1 else cd1) l1 ∧
                           TypeSafe ev1 (Accounts st1) (Stack st1) (Memory st1) (Storage st1) cd1 ∧
                           fullyInitialised ev1 (Accounts st1) (Stack st1) ⟶
                           (xe1 ≠ Nil ⟶ (∃len arr. (t1 = MTArray len arr) ∧
                                        (case t1' of MTValue val ⇒ CompMemType (if c1 then Memory
st1 else cd1) len arr t1' l1 l1'
                                                                |(MTArray l' ar') ⇒ (∃p. accessStore l1' (if
c1 then Memory st1 else cd1) = Some (MPointer p) ∧
                                                                                    CompMemType (if c1 then
Memory st1 else cd1) len arr t1' l1 p))))"
  "∀l2' v2' t2' g2'. ssel t2 l2 xe2 ev2 cd2 st2 g2 = Normal ((l2', t2'), g2') ∧
                           SCon t2 l2 (Storage st2 (Address ev2)) ∧
                           TypeSafe ev2 (Accounts st2) (Stack st2) (Memory st2) (Storage st2) cd2 ∧
                           fullyInitialised ev2 (Accounts st2) (Stack st2) ⟶
                           (xe2 ≠ Nil ⟶ CompStoToType t2 t2' l2 l2')"
  "∀v t g4'. expr e4 ev4 cd4 st4 g4 = Normal ((v, t), g4') ∧
                TypeSafe ev4 (Accounts st4) (Stack st4) (Memory st4) (Storage st4 ) cd4 ∧
                  fullyInitialised ev4 (Accounts st4) (Stack st4) ⟶
                    (case t of Value typ ⇒ (typeCon typ (extractValueType(v)) ∧ (∃xx. v = KValue xx))
                     | Calldata struct ⇒ (MCon struct cd4 (extractValueType(v)) ∧ (∃xx. v = KCDptr xx)
                                          ∧ (∃stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈|
fmran (Denvalue ev4)∧ accessStore stloc (Stack st4) =Some (KCDptr p)
                                          ∧ ((tp'' = struct ∧ v = KCDptr p) ∨
```

```
                                                (∃len arr. (extractValueType v) ≠ p ∧ tp'' = MTArray len
arr ∧ CompMemType cd4 len arr struct p (extractValueType v)))))
                        | type.Memory struct ⇒ (MCon struct (Memory st4) (extractValueType(v)) ∧ (∃xx. v
= KMemptr xx) ∧
                                                (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran
(Denvalue ev4)∧ accessStore stloc (Stack st4) =Some (KMemptr p)
                                                ∧
                                                    ((tp'' = struct ∧ v = (KMemptr p)) ∨ (∃len arr.
(extractValueType v)
                                                        ≠ p ∧tp'' = MTArray len arr ∧ CompMemType
(Memory st4) len arr struct
                                                        p (extractValueType v)))))
                        | type.Storage struct ⇒ (SCon struct (extractValueType(v)) (Storage st4 (Address
ev4))) ∧ (∃xx. v = KStoptr xx)
                                                  ∧ (∃stloc tp''. (type.Storage tp'', Storeloc stloc) |∈|
fmran (Denvalue ev4)
                                                  ∧((((tp'' = struct ∧ v = KStoptr stloc) ∨
                                                                ((extractValueType v) ≠ stloc ∧
CompStoType tp'' struct stloc (extractValueType v))))
                                                                                    )))"

"∀ev cd k m g'. load lcp lis lxs lev0 lcd0 lk lm lev lcd lst lg = Normal ((ev, cd, k, m), g') ∧
                        (¬lcp ⟶ (∀locs tp. MCon tp (Memory lst) locs ⟶ MCon tp lm locs)
                                        ∧ Toploc (Memory lst) ≤ Toploc lm
                                        ∧ ncpDenvalueLimit lev0 lev lk (Stack lst) (Memory lst)
                                        ∧ ncpOMemInDMem (Memory lst) lm
                                        ∧ ncpElementsNoSubPref (Memory lst) lm
                                        ∧ ncpNewSelfPoint (Memory lst) lm
∧ Address lev = Address lev0
) ∧
                    (TypeSafe lev (Accounts lst) (Stack lst) (Memory lst) (Storage lst ) lcd) ∧
                    (TypeSafe lev0 (Accounts lst) lk lm (Storage lst ) lcd0) ∧
                     fullyInitialised lev (Accounts lst) (Stack lst) ∧ fullyInitialised lev0 (Accounts
lst) lk
⟶
                    (TypeSafe ev (Accounts lst) k m (Storage lst ) cd) ∧
 fullyInitialised ev (Accounts lst) k
∧ (¬lcp ⟶ (∀locs tp. MCon tp (Memory lst) locs ⟶ MCon tp m locs)
                                        ∧ Toploc (Memory lst) ≤ Toploc m
                                        ∧ ncpDenvalueLimit ev lev k (Stack lst) (Memory lst)
                                        ∧ ncpOMemInDMem (Memory lst) m
                                        ∧ ncpElementsNoSubPref (Memory lst) m
                                        ∧ ncpNewSelfPoint (Memory lst) m)"
"∀v3' t3' g3'. rexp l3 ev3 cd3 st3 g3 = Normal ((v3', t3'), g3') ∧ TypeSafe ev3 (Accounts st3) (Stack
st3) (Memory st3) (Storage st3 ) cd3
                                          ∧ fullyInitialised ev3 (Accounts st3) (Stack st3)⟶
                     (case t3' of Value typ ⇒ (typeCon typ (extractValueType(v3')) ∧ (∃xx. v3' =
KValue xx))
                                        | Calldata struct ⇒ (MCon struct cd3 (extractValueType(v3'))
∧ (∃xx. v3' = KCDptr xx)
                                            ∧ (∃stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈|
fmran (Denvalue ev3)∧ accessStore stloc (Stack st3) =Some (KCDptr p)
                                            ∧ ((tp'' = struct ∧ v3' = KCDptr p) ∨
                                            (∃len arr. (extractValueType v3') ≠ p ∧ tp'' = MTArray
len arr ∧ CompMemType cd3 len arr struct p (extractValueType v3'))))))
                                        | type.Memory struct ⇒ (MCon struct (Memory st3)
(extractValueType(v3'))∧ (∃xx. v3' = KMemptr xx)
                                          ∧ (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈|
fmran (Denvalue ev3)∧ accessStore stloc (Stack st3) =Some (KMemptr p)
                                            ∧ ((tp'' = struct ∧ v3' = KMemptr p) ∨
                                            (∃len arr. (extractValueType v3') ≠ p ∧ tp'' = MTArray
len arr ∧
                                                CompMemType (Memory st3) len arr struct p
(extractValueType v3'))))))
```

```
                                        | type.Storage struct ⇒ (SCon struct (extractValueType(v3'))
(Storage st3 (Address ev3))) ∧ (∃xx. v3' = KStoptr xx)
                                              ∧ (∃stloc tp''. (type.Storage tp'', Storeloc stloc) |∈|
fmran (Denvalue ev3)
                                              ∧(
                                                    (((tp'' = struct ∧ v3' = KStoptr
stloc) ∨
                                                    ((extractValueType v3') ≠ stloc ∧
CompStoType tp'' struct stloc (extractValueType v3'))))
                                                                        )))"
proof (induct rule: msel_ssel_expr_load_rexp.induct
    [where ?P1.0="λc1 t1 l1 xe1 ev1 cd1 st1 g1. (∀l1' t1' g1' arr. msel c1 t1 l1 xe1 ev1 cd1 st1 g1 =
Normal ((l1', t1'), g1')

                                       ∧ MCon t1 (if c1 then Memory st1 else cd1) l1
                                       ∧ TypeSafe ev1 (Accounts st1) (Stack st1) (Memory st1) (Storage
st1) cd1 ∧
                                          fullyInitialised ev1 (Accounts st1) (Stack st1)
                                        ⟶ (xe1 ≠ Nil ⟶
                                              (∃len arr. (t1 = MTArray len arr) ∧
                                                  (case t1' of MTValue val ⇒ CompMemType (if c1
then Memory st1 else cd1) len arr t1' l1 l1'
                                                    |(MTArray l' ar') ⇒ (∃p. accessStore
l1' (if c1 then Memory st1 else cd1) = Some (MPointer p) ∧
                                                                                CompMemType (if
c1 then Memory st1 else cd1) len arr t1' l1 p )))))
"
     and ?P2.0="λt2 l2 xe2 ev2 cd2 st2 g2. (∀l2' v2' t2' g2'. ssel t2 l2 xe2 ev2 cd2 st2 g2 = Normal
((l2', t2'), g2')∧ SCon t2 l2 (Storage st2 (Address ev2))
                                       ∧ TypeSafe ev2 (Accounts st2) (Stack st2) (Memory st2)
(Storage st2 ) cd2
                                       ∧ fullyInitialised ev2 (Accounts st2) (Stack st2) ⟶
                                         (xe2 ≠ Nil ⟶ CompStoType t2 t2' l2 l2'))"
     and ?P3.0="λe4 ev4 cd4 st4 g4. (∀v t g4'. expr e4 ev4 cd4 st4 g4 = Normal ((v, t), g4') ∧
                  TypeSafe ev4 (Accounts st4) (Stack st4) (Memory st4) (Storage st4 ) cd4 ∧
               fullyInitialised ev4 (Accounts st4) (Stack st4) ⟶
                  (case t of Value typ ⇒ (typeCon typ (extractValueType(v)) ∧ (∃xx. v = KValue xx))
                   | Calldata struct ⇒ (MCon struct cd4 (extractValueType(v)) ∧ (∃xx. v = KCDptr xx)
∧ (∃stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue ev4)∧ accessStore stloc (Stack
st4) =Some (KCDptr p)
                                         ∧ ((tp'' = struct ∧ v = KCDptr p) ∨
                                         (∃len arr. (extractValueType v) ≠ p ∧ tp'' = MTArray len
arr ∧ CompMemType cd4 len arr struct p (extractValueType v)))))
                   | type.Memory struct ⇒ (MCon struct (Memory st4) (extractValueType(v)) ∧ (∃xx. v
= KMemptr xx) ∧
                                       (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran
(Denvalue ev4)∧ accessStore stloc (Stack st4) =Some (KMemptr p)
                                         ∧
                                             ((tp'' = struct ∧ v = (KMemptr p)) ∨ (∃len arr.
(extractValueType v)
                                                 ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType
(Memory st4) len arr
                                                 struct p (extractValueType v)))))
                   | type.Storage struct ⇒ (SCon struct (extractValueType(v)) (Storage st4 (Address
ev4))) ∧ (∃xx. v = KStoptr xx)
                                         ∧ (∃stloc tp''. (type.Storage tp'', Storeloc stloc) |∈|
fmran (Denvalue ev4)
                                         ∧(
                                                 (((tp'' = struct ∧ v = KStoptr stloc)
∨
                                                 ((extractValueType v) ≠ stloc ∧
CompStoType tp'' struct stloc (extractValueType v))))
                                                                    ))))"
     and ?P4.0="λlcp lis lxs lev0 lcd0 lk lm lev lcd lst lg. (∀ev cd k m g'. load lcp lis lxs lev0
```

```
lcd0 lk lm lev lcd lst lg = Normal ((ev, cd, k, m), g') ∧
                                        (¬lcp ⟶ (∀ locs tp. MCon tp (Memory lst) locs ⟶ MCon tp
lm locs)
                                    ∧ Toploc (Memory lst) ≤ Toploc lm
                                    ∧ ncpDenvalueLimit lev0 lev lk (Stack lst) (Memory lst)
                                    ∧ ncpOMemInDMem (Memory lst) lm
                                    ∧ ncpElementsNoSubPref (Memory lst) lm
                                    ∧ ncpNewSelfPoint (Memory lst) lm
                                    ∧ Address lev = Address lev0
                                ) ∧
                (TypeSafe lev (Accounts lst) (Stack lst) (Memory lst) (Storage lst ) lcd) ∧
                (TypeSafe lev0 (Accounts lst) lk lm (Storage lst ) lcd0) ∧
                fullyInitialised lev (Accounts lst) (Stack lst) ∧ fullyInitialised lev0 (Accounts
lst) (lk) ⟶
                                                                (TypeSafe ev (Accounts lst) k m (Storage
lst ) cd) ∧

 fullyInitialised ev (Accounts lst) (k) ∧
                    (¬lcp ⟶ (∀ locs tp. MCon tp (Memory lst) locs ⟶ MCon tp m locs)
                                    ∧ Toploc (Memory lst) ≤ Toploc m
                                    ∧ ncpDenvalueLimit ev lev k (Stack lst) (Memory lst)
                                    ∧ ncpOMemInDMem (Memory lst) m
                                    ∧ ncpElementsNoSubPref (Memory lst) m
                                    ∧ ncpNewSelfPoint (Memory lst) m)
)"
      and ?P5.0="λl3 ev3 cd3 st3 g3. (∀ v3' t3' g3'. rexp l3 ev3 cd3 st3 g3 = Normal (( v3', t3'), g3')
∧ TypeSafe ev3 (Accounts st3) (Stack st3) (Memory st3) (Storage st3 ) cd3 ∧
                fullyInitialised ev3 (Accounts st3) (Stack st3)⟶
                   (case t3' of Value typ ⇒ (typeCon typ (extractValueType(v3')) ∧ (∃ xx. v3' =
KValue xx))
                                    | Calldata struct ⇒ (MCon struct cd3 (extractValueType(v3'))
∧ (∃ xx. v3' = KCDptr xx)
                                        ∧ (∃ stloc tp'' p. (Calldata tp'', Stackloc stloc) |∈|
fmran (Denvalue ev3)∧ accessStore stloc (Stack st3) =Some (KCDptr p)
                                        ∧ ((tp'' = struct ∧ v3' = KCDptr p) ∨
                                        (∃ len arr. (extractValueType v3') ≠ p ∧ tp'' = MTArray
len arr ∧ CompMemType cd3 len arr struct p (extractValueType v3'))))
                                    )
                                    | type.Memory struct ⇒ (MCon struct (Memory st3)
(extractValueType(v3'))∧ (∃ xx. v3' = KMemptr xx)
                                     ∧ (∃ stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈|
fmran (Denvalue ev3)∧ accessStore stloc (Stack st3) =Some (KMemptr p)
                                        ∧ ((tp'' = struct ∧ v3' = KMemptr p) ∨
                                        (∃ len arr. (extractValueType v3') ≠ p ∧ tp'' = MTArray
len arr ∧
                                            CompMemType (Memory st3) len arr struct p
(extractValueType v3'))))
                                    | type.Storage struct ⇒ (SCon struct (extractValueType(v3'))
(Storage st3 (Address ev3))) ∧ (∃ xx. v3' = KStoptr xx)
                                        ∧ (∃ stloc tp''. (type.Storage tp'', Storeloc stloc) |∈|
fmran (Denvalue ev3)
                                        ∧(
                                                ((tp'' = struct ∧ v3' = KStoptr
stloc) ∨
                                                ((extractValueType v3') ≠ stloc ∧
CompStoType tp'' struct stloc (extractValueType v3'))))
                                                                ))))"
      ])
  case (1 _ _ _ _ _ _ g)
  then show ?case using msel.simps(1) by simp
next
  case (2 _ _ _ _ _ _ _ g)
  then show ?case using msel.simps(2) by simp
next
```

```
  case (3 mm al t loc x env cd st g)
  show ?case
  proof(intros)
    fix l1' t1' g1' arr
    assume a1:
      "local.msel mm (MTArray al t) loc [x] env cd st g = Normal ((l1', t1'), g1') ∧

       MCon (MTArray al t) (if mm then Memory st else cd) loc ∧ TypeSafe env (Accounts st) (Stack st)
(Memory st) (Storage st) cd
      ∧ fullyInitialised env (Accounts st) (Stack st)"

    then obtain kv b g4' where a20: "local.expr x env cd st g = Normal ((KValue kv, Value (TUInt b)),
g4')"
      and a30: "less (TUInt b) (TUInt b256) kv (ShowL_int (int al)) = Some ((ShowL_bool True), TBool)"
      using a1 msel.simps(3)
      by (simp split: result.split_asm prod.split_asm types.split_asm type.split_asm if_split_asm
stackvalue.split_asm option.split_asm memoryvalue.split_asm)
    have a50: "checkUInt b kv" using 3(1) a1 a20 typeCon.simps(2)[of b "extractValueType (KValue kv)"]
extractValueType.simps(1)[of kv] by auto
    then have a60:"ReadL_int(kv) < int al " using a20 a30 less_def plift.simps(2)[of "(<)" b b256 kv
       "(ShowL_int (int al))"] Read_ShowL_id[of "(int al)"] unfolding createBool_def
       ShowL_bool_def by (simp split:if_split_asm)
    then have a62:"0≤ReadL_int(kv)" and a65:"(ShowL_int (ReadL_int kv)) = kv" using a50 checkUInt_def[of
b kv] by simp+
    have a64:"(ShowL_nat (nat (ReadL_int kv))) = kv" using ShowLnat_ReadLint_inverse a62 a50 unfolding
checkUInt_def by simp
    then have a70:"(nat (ReadL_int kv)) ∈ {0..al-1}" using a60 a62 by simp
    have a90: "l1' = (hash loc kv)" using a20 msel.simps(3)[of mm al t loc x env cd st g] a30 a1 by
simp
    then have a100: "t = t1'" using a1 msel.simps by (simp split:if_split_asm result.split_asm
prod.split_asm stackvalue.split_asm type.split_asm)
    have **:"MCon (MTArray al t) (if mm then Memory st else cd) loc" using a1 by blast
    then have defexp:"(if al = 0 then False else
                        ∀i::nat <al.
                                              (case (accessStore (hash loc (ShowL_nat i))((if mm then
Memory st else cd))) of
                                               (Some (MPointer loc2)) ⇒
                                                     (case t of MTArray len' arr' ⇒ (MCon t (if mm
then Memory st else cd) (loc2))
                                                         | _ ⇒ False)
                                                | Some(MValue val) ⇒
                                                     (case t of MTValue typ ⇒ (MCon t (if mm then
Memory st else cd) (hash loc (ShowL_nat i)))
                                                         | _ ⇒ False)
                                                | _ ⇒ False
                                              )
          ∧ ((∃p. accessStore loc (if mm then Memory st else cd) = Some (MPointer p)) ∨ accessStore loc
(if mm then Memory
               st else cd) = None))" using MCon.simps(2)[of al t "(if mm then Memory st else cd)" loc ]
by simp
    have alNonZero:"al ≠ 0" using a60 a62 by simp
    then have b0:"(case (accessStore (hash loc (ShowL_nat (nat (ReadL_int kv))))((if mm then Memory st
else cd))) of
                                             (Some (MPointer loc2)) ⇒
                                                   (case t of MTArray len' arr' ⇒ (MCon t (if mm then
Memory st else cd) (loc2))
                                                       | _ ⇒ False)
                                              | Some(MValue val) ⇒
                                                   (case t of MTValue typ ⇒ (MCon t (if mm then
Memory st else cd) (hash loc
                                                       (ShowL_nat (nat (ReadL_int kv)))))
                                                       | _ ⇒ False)
                                              | _ ⇒ False
                                             )" using defexp a60 a65 alNonZero by simp
```

```
    then obtain v1' where b1:"(accessStore (hash loc (ShowL_nat (nat (ReadL_int kv))))((if mm then
Memory st else cd))) = Some v1'" using a64
      by fastforce
    then have a80:"(accessStore (hash loc kv) (if (¬mm) then cd else (Memory st))) = Some(v1')" using
a1 a70 a64 by auto


    assume b99:" [x] ≠ []"
    have "∃len arr.
          MTArray al t = MTArray len arr ∧
          (case t1' of
           MTArray l' ar' ⇒
             ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
                 CompMemType (if mm then Memory st else cd) len arr t1' loc p
           | MTValue val ⇒
               CompMemType (if mm then Memory st else cd) len arr t1' loc l1')"
    proof(cases t)
      case (MTArray x11 x12)
      then obtain p where "accessStore (hash loc kv) (if mm then Memory st else cd) = Some (MPointer
p)" using b0 b1 a1 a70 a65 a80 a64 defexp
        by (metis (no_types, lifting) mtypes.simps(5) memoryvalue.exhaust memoryvalue.simps(5)
Option.option.simps(5))
      moreover have "CompMemType (if mm then Memory st else cd) al t t loc p" using calculation
        using MTArray a60 a62 a80 b1 by force
      ultimately have "∃p. accessStore (hash loc kv) (if mm then Memory st else cd) = Some (MPointer
p) ∧ (CompMemType (if mm then Memory st else cd) al t t loc p)" by simp
      then show ?thesis using MTArray
        using "**" CompTypeRemainsMCon mtypes.simps(5) a100 a90 by blast
    next
      case (MTValue x2)

      then show ?thesis
        using MTValue CompMemType.simps(1)
        using a100 a60 a62 a64 a90 nat_less_iff
        by (metis (lifting) mtypes.simps(6))

    qed
    then show "∃len arr.
          MTArray al t = MTArray len arr ∧
          (case t1' of
           MTArray l' ar' ⇒
             ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
                 CompMemType (if mm then Memory st else cd) len arr t1' loc p
           | MTValue val ⇒
               CompMemType (if mm then Memory st else cd) len arr t1' loc l1')"
      using a90 a100 by blast
  qed
next
  case (4 mm al t loc x y ys env cd st g)
  show ?case
  proof (intros)
    fix l1' t1' g1' arr
    assume a1: "local.msel mm (MTArray al t) loc (x # y # ys) env cd st g = Normal ((l1', t1'), g1') ∧
       MCon (MTArray al t) (if mm then Memory st else cd) loc ∧ TypeSafe env (Accounts st) (Stack st)
(Memory st) (Storage st) cd
        ∧ fullyInitialised env (Accounts st) (Stack st)"

    then obtain kv b g4' l where a20: "local.expr x env cd st g = Normal ((KValue kv, Value (TUInt
b)), g4')"
      and a30: "less (TUInt b) (TUInt b256) kv (ShowL_int (int al)) = Some ((ShowL_bool True), TBool)"
      and a40: "accessStore (hash loc kv) (if mm then Memory st else cd) = Some (MPointer l)"
      and a45:"local.msel mm (MTArray al t) loc (x # y # ys) env cd st g = local.msel mm t l (y # ys)
env cd st g4'"
      using a1 msel.simps(4) by (simp split: result.split_asm prod.split_asm types.split_asm
```

```
type.split_asm if_split_asm stackvalue.split_asm option.split_asm memoryvalue.split_asm)

    have a50: "checkUInt b kv" using 4(1) a1 a20 by simp
    then have a60:"ReadL_int(kv) < int al " using a20 a30 less_def plift.simps(2)[of "(<)" b b256 kv
        "(ShowL_int (int al))"] Read_ShowL_id[of "int al"] unfolding createBool_def
        ShowL_bool_def by (simp split:if_split_asm)
    then have a62:"0≤ReadL_int(kv)" and a65:"(ShowL_int (ReadL_int kv)) = kv" using a50 checkUInt_def[of
b kv] by simp+
    then have a64:"(ShowL_nat (nat (ReadL_int kv))) = kv" using ShowLnat_ReadLint_inverse a50 unfolding
checkUInt_def by simp
    then have a70:"(nat (ReadL_int kv)) ∈ {0..al-1}" using a60 by simp
    then have a80:"(accessStore (hash loc kv) (if (¬mm) then cd else (Memory st))) = Some(MPointer l)"
using a1(1) a40 by presburger
    have a90:"al > 0" using a60 a62 by simp
    have a91:" MCon (MTArray al t) (if mm then Memory st else cd) loc" using a1 by blast
    then have defexp:"(case (accessStore (hash loc (ShowL_nat (nat (ReadL_int kv))))((if mm then Memory
st else cd))) of
                                        (Some (MPointer loc2)) ⇒
                                              (case t of MTArray len' arr' ⇒ (MCon t (if mm then
Memory st else cd) (loc2))
                                                  | _ ⇒ False)
                                         | Some(MValue val) ⇒
                                              (case t of MTValue typ ⇒ (MCon t (if mm then Memory
st else cd) (hash loc
                                                  (ShowL_nat (nat (ReadL_int kv)))))
                                                  | _ ⇒ False)
                                         | _ ⇒ False
                                        )" using MCon.simps(2)[of al t _ loc] a90 a60 by simp
    then have a95:"case t of MTArray len' arr' ⇒ MCon t (if mm then Memory st else cd) l | MTValue
Types ⇒ False" using a80 a64 by (auto split:if_splits )
    then obtain len' arr' where tdef:"t = MTArray len' arr'" using a1 a70 a65 a64 a80 a90 a60 defexp
by (cases t; simp)
    then have a100:"MCon t (if mm then Memory st else cd) l" using a1 a70 a65 a64 a80 a90 a60 defexp
a95 by simp


    assume b99:"x # y # ys ≠ []"
    have b10: "∃len arr.
          MTArray al t = MTArray len arr ∧
          (case t1' of
           MTArray l' ar' ⇒
             ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
                 CompMemType (if mm then Memory st else cd) len arr t1' loc p
           | MTValue val ⇒
              CompMemType (if mm then Memory st else cd) len arr t1' loc l1')"
    proof(cases ys)
      case Nil
      then have b15:"∀l1' v1' t1' g1' arr.
       local.msel mm t l (y # ys) env cd st g4' = Normal ((l1', t1'), g1') ∧
       MCon t (if mm then Memory st else cd) l ∧ TypeSafe env (Accounts st) (Stack st) (Memory st)
(Storage st) cd ⟶

        (y # ys ≠ [] ⟶
         (∃len arr.
            t = MTArray len arr ∧
            (case t1' of MTArray l' ar' ⇒ ∃p. accessStore l1' (if mm then Memory st else cd) = Some
(MPointer p)
                                          ∧ CompMemType (if mm then Memory st else cd) len arr t1' l
p

             | MTValue val ⇒ CompMemType (if mm then Memory st else cd) len arr t1' l l1')))"
       using 4(2)[of "((KValue kv, Value (TUInt b)))" g4' "(kv, b)" g4' kv b ] a20 a30 a40 a45 a100 a1
by (auto split:if_splits option.splits )
```

```
      have b20: "local.msel mm t l [y] env cd st g4' = Normal ((l1', t1'), g1')" using a45 Nil a1 by
simp
      then obtain kv' b' where b30:"expr y env cd st g4' = Normal (((KValue kv', Value (TUInt b')),
g1'))"
        using msel.simps(3)[of mm len' arr' l y env cd st g4'] tdef by (auto split:result.splits
stackvalue.splits type.splits types.splits if_splits)
      then have b40:"Valuetypes.less (TUInt b') (TUInt b256) kv' (ShowL_int (int len')) = Some (ShowL_bool
True, TBool)"
        using msel.simps(3)[of mm len' arr' l y env cd st g4'] tdef b20 by (auto split:result.splits
stackvalue.splits type.splits types.splits if_splits)

      then have b50:"l1' = hash l kv'"
        using msel.simps(3)[of mm len' arr' l y env cd st g4'] tdef b20 b30
        by (auto split:result.splits stackvalue.splits type.splits types.splits if_splits)
      then have b60:"ReadL_int(kv') < int len'"
        using tdef b20 b40 less_def plift.simps(2)[of "(<)" b' b256 kv' ] Read_ShowL_id[of "int len'"]
createBool_def ShowL_bool_def
        by (auto split:result.splits stackvalue.splits type.splits types.splits if_splits)
      then have "(y # ys ≠ [] ⟶
        (∃len arr.
           t = MTArray len arr ∧
           (case t1' of
            MTArray l' ar' ⇒
              ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
                  CompMemType (if mm then Memory st else cd) len arr t1' l p
            | MTValue val ⇒
                CompMemType (if mm then Memory st else cd) len arr t1' l l1')))"
        using b15 b20 a1 a100 Nil by simp
      then have b70:"(
           (case t1' of MTArray l' ar' ⇒ ∃p. accessStore l1' (if mm then Memory st else cd) = Some
(MPointer p)
                                                ∧ CompMemType (if mm then Memory st else cd) len' arr'
t1' l p

            | MTValue val ⇒ CompMemType (if mm then Memory st else cd) len' arr' t1' l l1'
                          ))"
        using b99 tdef by simp
      then show ?thesis
      proof(cases t1')
        case (MTArray x11 x12)
        then have "∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p)
                   ∧ CompMemType (if mm then Memory st else cd) len' arr' t1' l p
                   "
          using b70 by simp
        then obtain p where pdef:" accessStore l1' (if mm then Memory st else cd) = Some (MPointer p)
∧ CompMemType (if mm then Memory st else cd) len' arr' t1' l p" by auto
        have "CompMemType (if mm then Memory st else cd) al ( MTArray len' arr') t1' loc p"
          using CompMemType.simps(2)[of "(if mm then Memory st else cd)" al len' arr' t1' loc p]
          by (metis a40 a60 a62 a64 nat_less_iff pdef)
        then show ?thesis using MTArray b70 pdef tdef
          using CompTypeRemainsMCon mtypes.simps(5) a91 by blast
      next
        case (MTValue x2)
        then have c10:"CompMemType (if mm then Memory st else cd) len' arr' t1' l l1'" using b70 by
simp
        then have noteq:"l ≠ l1'"
          by (metis b50 hash_inequality)
        have "l ≠ loc" using b50 MConPtrsMustBeSubLocs_CompMemType[of al t "(if mm then Memory st else
cd)" loc] a91
          using MConSubTypes a100 CompMemJustType.simps(2) tdef by blast
        then show ?thesis using MTValue b70 tdef using a1 a70 a65 a64 a80 a90 a60 defexp a95 noteq a40
c10
          by (metis (no_types, lifting) mtypes.simps(6) a62 nat_less_iff
              CompMemType.simps(2))
```

```
      qed
    next
      case (Cons a list)
      then have b15:"∀l1' v1' t1' g1' arr.
       local.msel mm t l (y # ys) env cd st g4' = Normal ((l1', t1'), g1') ∧
       MCon t (if mm then Memory st else cd) l ∧ TypeSafe env (Accounts st) (Stack st) (Memory st)
(Storage st) cd ⟶
       (y # ys ≠ [] ⟶(∃len arr.
            t = MTArray len arr ∧
            (case t1' of MTArray l' ar' ⇒ ∃p. accessStore l1' (if mm then Memory st else cd) = Some
(MPointer p)
                                          ∧ CompMemType (if mm then Memory st else cd) len arr t1' l
p

             | MTValue val ⇒ CompMemType (if mm then Memory st else cd) len arr t1' l l1' )))"
        using 4(2)[of "((KValue kv, Value (TUInt b)))" g4' "(kv, b)" g4' kv b ] a20 a30 a40 a45 a1 by
(auto split:if_splits option.splits )
      have b20: "local.msel mm t l (y#ys) env cd st g4' = Normal ((l1', t1'), g1')" using a45 Cons a1
by simp
      then have b30:"∃len arr.
            t = MTArray len arr ∧
            (case t1' of MTArray l' ar' ⇒ ∃p. accessStore l1' (if mm then Memory st else cd) = Some
(MPointer p)
                                          ∧ CompMemType (if mm then Memory st else cd) len arr t1' l
p

             | MTValue val ⇒ CompMemType (if mm then Memory st else cd) len arr t1' l l1' )"
        using b15 b20 a1 a100 by simp
      then show ?thesis
      proof(cases t1')
        case (MTArray x11 x12)
        then have " ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
CompMemType (if mm then Memory st else cd) al t t1' loc p"
          using b30 a40 a62 a64 nat_less_iff
          using a60 by fastforce
        then show ?thesis using MTArray
          using CompTypeRemainsMCon mtypes.simps(5) a91 by blast
      next
        case (MTValue x2)
        then have "CompMemType (if mm then Memory st else cd) al t t1' loc l1'" using b30
          using a40 a60 a62 a64 nat_less_iff by fastforce
        then show ?thesis using MTValue b30 by force
      qed
    qed
    then show "∃len arr.
          MTArray al t = MTArray len arr ∧
          (case t1' of
           MTArray l' ar' ⇒
             ∃p. accessStore l1' (if mm then Memory st else cd) = Some (MPointer p) ∧
                 CompMemType (if mm then Memory st else cd) len arr t1' loc p
           | MTValue val ⇒
              CompMemType (if mm then Memory st else cd) len arr t1' loc l1')" by simp
  qed
next
  case (5 tp loc vi vj vk g)
  then show ?case
  proof (intros)
    fix l2' v2' t2' g2'
    assume a1: "local.ssel tp loc [] vi vj vk g = Normal ((l2', t2'), g2') ∧
       SCon tp loc (Storage vk (Address vi)) ∧ TypeSafe vi (Accounts vk) (Stack vk) (Memory vk)
(Storage vk) vj
∧ fullyInitialised vi (Accounts vk) (Stack vk)"
    assume *:"[] ≠ []"
    then have a10:"loc = l2'" and a20:"t2' = tp" using ssel.simps(1) by simp+
```

```
    show " CompStoType tp t2' loc l2' " using * by simp
  qed
next
  case (6 _ _ _ _ _ _ _ g)
  then show ?case using ssel.simps(2) by simp
next
  case (7 al t loc x xs env cd st g)
  show ?case
  proof(intros)
    fix l2' v2' t2' g2'
    assume a1:"local.ssel (STArray al t) loc (x # xs) env cd st g = Normal ((l2', t2'), g2') ∧
       SCon (STArray al t) loc (Storage st (Address env)) ∧ TypeSafe env (Accounts st) (Stack st)
(Memory st) (Storage st) cd
      ∧ fullyInitialised env (Accounts st) (Stack st)"
    assume b99:"x # xs ≠ []"
    then obtain kv b g4' where a20: "local.expr x env cd st g = Normal ((KValue kv, Value (TUInt b)),
g4')"
      and a30: "less (TUInt b) (TUInt b256) kv (ShowL_int (int al)) = Some (ShowL_bool True, TBool)"
      and a40:"ssel (STArray al t) loc (x # xs) env cd st g = ssel t (hash loc kv) xs env cd st g4'"
      using a1 ssel.simps(3) by (simp split: result.split_asm prod.split_asm types.split_asm
type.split_asm if_split_asm stackvalue.split_asm)

    have a50: "checkUInt b kv" using 7(1) a1 a20 by simp
    then have a60:"ReadL_int(kv) < int al " using a20 a30 less_def plift.simps(2)[of "(<)" b b256 kv
        "(ShowL_int (int al))"] Read_ShowL_id[of "(int al)"] unfolding createBool_def
        ShowL_bool_def by (simp split:if_split_asm)
    then have a62:"0≤ReadL_int(kv)" and a65:"(ShowL_int (ReadL_int kv)) = kv" using a50 checkUInt_def[of
b kv] by simp+
    then have a67:"(ShowL_nat (nat (ReadL_int kv))) = kv" using ShowLnat_ReadLint_inverse a50 unfolding
checkUInt_def by simp
    then have a68:"nat (ReadL_int kv) < al" by (simp add: a60 a62 nat_less_iff)
    then have b10:"SCon t (hash loc kv) (Storage st (Address env)) ∧ (nat (ReadL_int kv)) ∈ {0..al-1}"
      using a1 a65 a67 by force

    have b10:"CompStoType (STArray al t) t2' loc l2' ∧ loc ≠ l2'"
    proof(cases xs)
      case Nil
      then have "ssel t (hash loc kv) xs env cd st g4' = Normal(((hash loc kv), t),g2')" using
ssel.simps(1) a1 a40 by simp
      then have b30:"l2' = (hash loc kv) ∧ t2' = t" using a40 a1 by simp
      then have "(∃i<al. TypedStoSubpref l2' (hash loc (ShowL_nat i)) t)" using a67 a68
TypedStoSubpref_sameLoc by auto
      moreover have "(∃i<al. CompStoType t t2' (hash loc (ShowL_nat i)) l2')" using a67 a68 b30
CompStoType_sameLocNdTyp by auto
      moreover have "l2' ≠ loc" using b30 by (simp add: hash_inequality)
      ultimately show ?thesis using TypedStoSubpref.simps(2)[of l2' loc al t ] b30
CompStoType.simps(2)[of al t t2' loc l2'] by simp

    next
      case (Cons a list)
      have b20:"xs ≠ [] ⟶ CompStoType t t2' (hash loc kv) l2'"
        using 7(2) a1 a20 a30 a40 b10 by simp
      then have b30:"CompStoType t t2' (hash loc kv) l2'" using Cons by simp
      have "al > 0" using a60 by (metis a62 bot_nat_0.not_eq_extremum less_nat_zero_code nat_less_iff)
      then have "∀i::nat. i∈{0..al - 1} ⟶ i<al" using a60 by auto
      then have b40:"(∃i<al. TypedStoSubpref l2' (hash loc (ShowL_nat i)) t ∨ l2' = hash loc (ShowL_nat
i))"
        using a60 a65 a67 b20 b30 by (metis CompStoType_imps_TypedStoSubpref a68)

      then show ?thesis
      proof(cases t)
        case (STArray x11 x12)
        then have c1:"loc ≠ l2'" using b30 TypedStoSubpref.simps(2)[of l2' "(hash loc kv)" x11 x12]
TypedStoSubpref_hashes
```

```
        by (metis b40 hash_inequality)
      have "((∃ i<al. CompStoType t t2' (hash loc (ShowL_nat i)) l2'))" using b30 a67 a68 by auto
      moreover have "((∃ i<al. TypedStoSubpref l2' (hash loc (ShowL_nat i)) t))"
        using b30 a67 a68 TypedStoSubpref_sameLoc b40 by blast
      ultimately show ?thesis using CompStoType.simps(2)[of al t t2' loc l2'] b30
TypedStoSubpref.simps(2)[of l2' loc al t] c1 by simp
    next
      case (STMap x21 x22)
      then have c1:"loc ≠ l2'"
        using b30 TypedStoSubpref.simps(2) TypedStoSubpref_hashes by (metis b40)
      have "((∃ i<al. CompStoType t t2' (hash loc (ShowL_nat i)) l2'))" using b30 a67 a68 by auto
      moreover have "((∃ i<al. TypedStoSubpref l2' (hash loc (ShowL_nat i)) t))"
        using b30 a67 a68 TypedStoSubpref_sameLoc b40 by blast
      ultimately show ?thesis using CompStoType.simps(2)[of al t t2' loc l2'] b30
TypedStoSubpref.simps(2)[of l2' loc al t] c1 by simp
    next
      case (STValue x3)
      then have "l2' = (hash loc kv)" using b30 TypedStoSubpref.simps(1)[of l2' " (hash loc kv)" x3]
by auto
      then have c1:"loc ≠ l2'" by (metis hash_inequality)
      have "(∃ i<al. CompStoType t t2' (hash loc (ShowL_nat i)) l2')" using b30 a67 a68 by auto
      moreover have "(∃ i<al. TypedStoSubpref l2' (hash loc (ShowL_nat i)) t)" using b30 a67 a68
        using TypedStoSubpref_sameLoc b40 by blast
      ultimately show ?thesis using CompStoType.simps(2)[of al t t2' loc l2'] b30
TypedStoSubpref.simps(2)[of l2' loc al t] c1 by simp
    qed
  qed
  show "CompStoType (STArray al t) t2' loc l2'" using b10 by auto
 qed
next
 case (8 vs t loc x xs env cd st g)
 show ?case
 proof(intros)
  fix l2' v2' t2' g2'
  assume a1:"local.ssel (STMap vs t) loc (x # xs) env cd st g = Normal ((l2', t2'), g2') ∧
     SCon (STMap vs t) loc (Storage st (Address env)) ∧ TypeSafe env (Accounts st) (Stack st) (Memory
st) (Storage st) cd
    ∧ fullyInitialised env (Accounts st) (Stack st) "

  then obtain kv g4' t'' where a20: "local.expr x env cd st g = Normal ((KValue kv, Value t''),
g4')"
    and a30: "comp t'' vs"
    and a40:"ssel (STMap vs t) loc (x # xs) env cd st g = ssel t (hash loc kv) xs env cd st g4'"
    using a1 ssel.simps(4) by (simp split: result.split_asm prod.split_asm types.split_asm
type.split_asm if_split_asm stackvalue.split_asm)

  then have "typeCon t'' kv" using 8(1) a1 by simp
  then have b20:"typeCon vs kv" using SameCompTypeCon[of t'' vs kv] a30 by simp
  then have b10:"SCon t (hash loc kv) (Storage st (Address env))" using SCon.simps(3)[of vs t loc
"(Storage st (Address env))"] a1 by simp

  assume "x # xs ≠ []"
  then have b10:"CompStoType (STMap vs t) t2' loc l2'"
  proof(cases xs)
    case Nil
    then have "ssel t (hash loc kv) xs env cd st g4' = Normal(((hash loc kv), t),g2')" using
ssel.simps(1) a1 a40 by simp
    then have b30:"l2' = (hash loc kv) ∧ t2' = t" using a40 a1 by simp

    then have " (∃ i. typeCon vs i ∧ (TypedStoSubpref l2' (hash loc i) t ∨ l2' = hash loc i))" using
b20 by auto
    then show ?thesis using TypedStoSubpref.simps(3)[of l2' loc vs t ] b30 b20
      using CompStoType_sameLocNdTyp TypedStoSubpref_sameLoc by auto
  next
```

```
      case (Cons a list)
      have "(xs ≠ [] ⟶ CompStoType t t2' (hash loc kv) l2')" using 8(2) a1 a20 a30 a40 b10 by simp
      then have b30:"CompStoType t t2' (hash loc kv) l2'" using Cons by blast
      then show ?thesis
      proof(cases "t2' = t")
        case True
        then show ?thesis using b20 b30 a40 a1 Cons by auto
      next
        case False
        then have " (∃i. typeCon vs i ∧ CompStoType t t2' (hash loc i) l2')" using b20 b30 by auto
        then show ?thesis using b30 b20 using CompStoType.simps(3)[of vs t t2' loc l2'] False by auto
      qed
    qed
    show "CompStoType (STMap vs t) t2' loc l2'" using b10 by simp
  qed
next
  case (9 b x ev cd st g')
  then show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (e.INT b x) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st ) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st) "
    show " exprResultCon ev cd st v t "
    proof -
      have a10:"t = Value (TSInt b)" using expr.simps(1) a1 by (simp split:if_split_asm)
      then have a20:"v = KValue (createSInt b x)" using expr.simps(1) a1 by (simp split:if_split_asm)
      moreover have "typeCon (TSInt b) (createSInt b x) = checkSInt (b) (createSInt b x)" by simp
      moreover have "checkSInt (b) (createSInt b x)" using checkSIntCreate by simp
      ultimately have "typeCon (TSInt b) (extractValueType v)" by (simp split:if_split_asm)
      then show ?thesis using a10 a20 by (simp split:if_split_asm)
    qed
  qed
next
  case (10 b x ev cd st g')
  then show ?case
  proof (intros)
    fix v t g
    assume a1: "local.expr (UINT b x) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st)
(Stack st) (Memory st) (Storage st ) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "v = KValue (createUInt b x)" using expr.simps(2) by (simp split:if_split_asm)
    moreover have "t = Value (TUInt b)" using a1 expr.simps(2) by (simp split:if_split_asm)
    moreover have "typeCon (TUInt b) (createUInt b x) = checkUInt (b) (createUInt b x)" using a1
expr.simps(2) by (simp split:if_split_asm)
    moreover have "checkUInt b (createUInt b x)" using checkUIntCreate by simp
    ultimately show "exprResultCon ev cd st v t " by simp
  qed
next
  case (11 ad ev cd st g')
  then show ?case
  proof (intros)
    fix v t g
    assume a1: "local.expr (ADDRESS ad) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st)
(Stack st) (Memory st) (Storage st ) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    have a10:"v = KValue (createAddress ad)" using a1 expr.simps(3) by (simp split:if_split_asm)
    then have a20:"t = Value TAddr" using a1 expr.simps(3) by (simp split:if_split_asm)
    then have "typeCon TAddr ad = checkAddress ad" by simp
    then show "exprResultCon ev cd st v t"
      using a10 a20 unfolding checkAddress_def
      using createAddressNoDots
        typeCon.simps(4)[of "(createAddress ad)"]
        extractValueType.simps(1)[of "createAddress ad"] by (simp add: checkAddress_def)
```

```
  qed
next
  case (12 ad ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (BALANCE ad) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st ) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then obtain adv where 3:"expr ad ev cd st (g' - costs_e (BALANCE ad) ev cd st) = Normal ((KValue
adv, Value TAddr), g)" using
        expr.simps(4) by (simp split:if_split_asm prod.split_asm result.split_asm stackvalue.split_asm
type.split_asm types.split_asm)
    moreover have a5:"expr (BALANCE ad) ev cd st g' = Normal ((KValue (Bal ((Accounts st) adv)), Value
(TUInt b256)), g)" using a1
        expr.simps(4) 3 by (simp split:if_split_asm prod.split_asm result.split_asm
stackvalue.split_asm type.split_asm types.split_asm)
    moreover have "v = KValue (Bal ((Accounts st) adv))" using a1 a5 by simp
    ultimately show "exprResultCon ev cd st v t"
      using a1 typeSafeAccounts[of "ev" "(Accounts st)" "(Stack st)" "(Memory st)" "(Storage st)" cd]
balanceTypes_def by simp
  qed
next
  case (13 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr THIS ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st ) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "v = KValue (Address ev)" using expr.simps(5) by (simp split:if_split_asm)
    moreover have "t = Value TAddr" using a1 expr.simps(5) by (simp split:if_split_asm)
    moreover have "typeCon TAddr (Address ev) = checkAddress (Address ev)" using expr.simps(5) by
(simp split:if_split_asm)
    ultimately show "exprResultCon ev cd st v t"
      unfolding checkAddress_def using a1 unfolding TypeSafe_def addressFormat_def by simp
  qed
next
  case (14 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr SENDER ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "v = KValue (Sender ev)" using expr.simps(6) by (simp split:if_split_asm)
    moreover have "t = Value TAddr" using a1 expr.simps(6) by (simp split:if_split_asm)
    moreover have "typeCon TAddr (Sender ev) = checkAddress (Sender ev)" using expr.simps(6) by (simp
split:if_split_asm)
    ultimately show "exprResultCon ev cd st v t"
      unfolding checkAddress_def using a1 unfolding TypeSafe_def addressFormat_def by simp
  qed
next
  case (15 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr VALUE ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st) cd
                ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "t = Value (TUInt b256)" using expr.simps(7) by (simp split:if_split_asm)
    moreover have "v = KValue(Svalue ev)" using a1 expr.simps(7) by (simp split:if_split_asm)
    ultimately show "exprResultCon ev cd st v t"
      using a1 typeSafeSvalue[of "ev" "(Accounts st)" "(Stack st)" "(Memory st)" "(Storage st)" cd]
```

```
svalueTypes_def by simp
  qed
next
  case (16 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr TRUE ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st) cd
               ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "t = Value TBool" using expr.simps(8) by (simp split:if_split_asm)
    moreover have "v = KValue (ShowL_bool True)" using a1 expr.simps(8) by (simp split:if_split_asm)
    moreover have "typeCon TBool (ShowL_bool True) = checkBool (ShowL_bool True)" using expr.simps(8) by
(simp split:if_split_asm)
    ultimately show "exprResultCon ev cd st v t"
      unfolding checkBool_def ShowL_bool_def by (simp split:if_split_asm)
  qed
next
  case (17 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr FALSE ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st) cd
               ∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have "t = Value TBool" using expr.simps(9) by (simp split:if_split_asm)
    moreover have "v = KValue (ShowL_bool False)" using a1 expr.simps(9) by (simp split:if_split_asm)
    moreover have "typeCon TBool (ShowL_bool False) = checkBool (ShowL_bool False)" using expr.simps(9)
by (simp split:if_split_asm)
    ultimately show "exprResultCon ev cd st v t"
      unfolding checkBool_def ShowL_bool_def by (simp split:if_split_asm)
  qed
next
  case (18 x ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (NOT x) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack st)
(Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    moreover have 1: "assert Gas (λg. g > costs_e (NOT x) ev cd st) g' = Normal ((), g')" using
expr.simps(10) a1 by (simp split:if_split_asm)
    moreover have 2: "modify (λg. g - costs_e (NOT x) ev cd st) g' = Normal ((), g' - costs_e (NOT x) ev
cd st)" using a1 by simp
    moreover obtain v'' g'' where 3:"expr x ev cd st (g' - costs_e (NOT x) ev cd st) = Normal ((KValue
v'', Value TBool), g'')" using
        a1 expr.simps(10) by (simp split:if_split_asm prod.split_asm result.split_asm
stackvalue.split_asm type.split_asm types.split_asm)
    ultimately have a5:"v'' = ShowL_bool True ∨ v'' = ShowL_bool False" using a1
        expr.simps(10) by (simp split:if_split_asm prod.split_asm result.split_asm stackvalue.split_asm
type.split_asm types.split_asm)
    then consider (T) "v'' = ShowL_bool True" |(F) "v'' = ShowL_bool False" by auto
    then show "exprResultCon ev cd st v t"
    proof(cases)
      case T
      then have a10:"expr FALSE ev cd st g'' = Normal ((v, t), g)" using 1 2 3 expr.simps(10)[of x ev
cd st g'] a1 by simp
      then have "t = Value TBool" using expr.simps(9) by (simp split:if_split_asm)
      moreover have "v = KValue (ShowL_bool False)" using a1 expr.simps(9) a10 by (simp
split:if_split_asm)
      moreover have "typeCon TBool (ShowL_bool False) = checkBool (ShowL_bool False)" using expr.simps(9)
by (simp split:if_split_asm)
      ultimately show ?thesis unfolding checkBool_def ShowL_bool_def by (simp split:if_split_asm)
    next
```

```
      case F
      then have a20:"expr TRUE ev cd st g'' = Normal ((v, t), g)" using 1 2 3 expr.simps(10)[of x ev
cd st g'] a1 ReadShow.true_neq_false by simp
      then have "t = Value TBool" using expr.simps(8) by (simp split:if_split_asm)
      moreover have "v = KValue (ShowL_bool True)" using a1 expr.simps(8) a20 by (simp
split:if_split_asm)
      moreover have "typeCon TBool (ShowL_bool True) = checkBool (ShowL_bool True)" using expr.simps(8)
by (simp split:if_split_asm)
      ultimately show ?thesis unfolding checkBool_def ShowL_bool_def by (simp split:if_split_asm)
    qed
  qed
next
  case (19 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (PLUS e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a2:"lift expr add e1 e2 ev cd st (g' - costs_e (PLUS e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(11) a1 by (simp split:if_split_asm)
    moreover obtain t' where "t = Value t'" using a2 by (auto split:result.splits)
    moreover obtain v' where "v = KValue v'" using a2 by (auto split:result.splits)
    ultimately show "exprResultCon ev cd st v t"
      using oliftSuccess[of "(+)" e1 e2 ev cd st "(g' - costs_e (PLUS e1 e2) ev cd st)" v' t' g] add_def
a1 a2 by simp
  qed
next
  case (20 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (MINUS e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a2:"lift expr sub e1 e2 ev cd st (g' - costs_e (MINUS e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(12) a1 by (simp split:if_split_asm)
    moreover obtain t' where "t = Value t'" using a2 by (auto split:result.splits)
    moreover obtain v' where "v = KValue v'" using a2 by (auto split:result.splits)
    ultimately show "exprResultCon ev cd st v t"
      using oliftSuccess[of "(-)" e1 e2 ev cd st "(g' - costs_e (MINUS e1 e2) ev cd st)" v' t' g]
sub_def a2 a1 by simp
  qed
next
  case (21 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (LESS e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a2:"lift expr less e1 e2 ev cd st (g' - costs_e (LESS e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(13) a1 by (simp split:if_split_asm)
    moreover obtain t' where "t = Value t'" using a2 by (auto split:result.splits)
    moreover obtain v' where "v = KValue v'" using a2 by (auto split:result.splits)
    ultimately show "exprResultCon ev cd st v t"
      using pliftSuccess[of "(<)" e1 e2 ev cd st "(g' - costs_e (LESS e1 e2) ev cd st)" v' t' g]
less_def by simp
  qed
next
  case (22 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
```

```
    assume a1: "expr (EQUAL e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a2:"lift expr equal e1 e2 ev cd st (g' - costs_e (EQUAL e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(14) a1 by (simp split:if_split_asm)
    moreover obtain t' where "t = Value t'" using a2 by (auto split:result.splits)
    moreover obtain v' where "v = KValue v'" using a2 by (auto split:result.splits)
    ultimately show "exprResultCon ev cd st v t"
      using pliftSuccess[of "(=)" e1 e2 ev cd st "(g' - costs_e (EQUAL e1 e2) ev cd st)" v' t' g]
equal_def by simp
  qed
next
  case (23 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (AND e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a10:"lift expr vtand e1 e2 ev cd st (g' - costs_e (AND e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(15) a1 by (simp split:if_split_asm)
    then obtain v1 t1 g'' where a20:"expr e1 ev cd st (g' - costs_e (AND e1 e2) ev cd st) =
Normal((KValue v1, Value t1), g'')" using
        lift_def by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
    then obtain v2 t2 g''' where a25:"expr e2 ev cd st g'' = Normal((KValue v2, Value t2), g''')"
using lift_def a10 by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
    then obtain v' t' where a30:"vtand t1 t2 v1 v2 = Some(v', t')" using a10 a20 a25 a1 lift_def by
(auto split:option.split_asm)
    then have a90:"KValue v' = v" and a95:"Value t' = t" using a30 a1 a10 a20 a25 by (auto
split:result.split_asm stackvalue.split_asm type.split_asm option.split_asm)
    then have a100:"(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' =
        TBool" using lift_def[of expr vtand e1 e2 ev cd st] expr.simps(15)[of e1 e2 ev cd st g]
vtandTypeOut a30 a1 a10 a20 a25 by simp
    then show "exprResultCon ev cd st v t"
      using a30 vtandTypeCon[of t1 t2 v1 v2 v' t'] a90 a95 a30 a1 a10 a20 a25 a100 by auto
  qed
next
  case (24 e1 e2 ev cd st g')
  show ?case
  proof (intros)
    fix v t g
    assume a1: "expr (OR e1 e2) ev cd st g' = Normal ((v, t), g) ∧ TypeSafe ev (Accounts st) (Stack
st) (Memory st) (Storage st) cd
∧ fullyInitialised ev (Accounts st) (Stack st)"
    then have a10:"lift expr vtor e1 e2 ev cd st (g' - costs_e (OR e1 e2) ev cd st) = Normal ((v, t),
g)" using expr.simps(16) a1 by (simp split:if_split_asm)
    then obtain v1 t1 g'' where a20:"expr e1 ev cd st (g' - costs_e (OR e1 e2) ev cd st) =
Normal((KValue v1, Value t1), g'')" using
        lift_def by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
    then obtain v2 t2 g''' where a25:"expr e2 ev cd st g'' = Normal((KValue v2, Value t2), g''')"
using lift_def a10 by (auto split:result.split_asm stackvalue.split_asm type.split_asm)
    then obtain v' t' where a30:"vtor t1 t2 v1 v2 = Some(v', t')" using a10 a20 a25 a1 lift_def by
(auto split:option.split_asm)
    then have a90:"KValue v' = v" and a95:"Value t' = t" using a30 a1 a10 a20 a25 by (auto
split:result.split_asm stackvalue.split_asm type.split_asm option.split_asm)
    then have a100:"(v' = ShowL_bool True ∨ v' = ShowL_bool False) ∧ t' =
        TBool" using lift_def[of expr vtand e1 e2 ev cd st] expr.simps(16)[of e1 e2 ev cd st g]
vtorTypeOut a30 a1 a10 a20 a25 by simp
    then show "exprResultCon ev cd st v t"
      using a30 checkBool_def createBool_def ShowL_bool_def a90 a95 a30 a1 a10 a20 a25 a100 by auto
  qed
next
  case (25 i ev cd st g')
  show ?case
```

```
  proof(intros)
    fix v t g4' assume a1:"local.expr (LVAL i) ev cd st g' = Normal ((v, t), g4')
                           ∧ TypeSafe ev (Accounts st) (Stack st) (Memory st) (Storage st) cd
                           ∧ fullyInitialised ev (Accounts st) (Stack st)"
    obtain v' where "assert Gas ((<) (costs_e (LVAL i) ev cd st)) g' = Normal (v', g')" using a1 25
expr.simps(17) extractValueType.simps by (simp split:if_split_asm option.splits prod.splits)
    moreover obtain v'' g'' where " modify (λg. g - costs_e (LVAL i) ev cd st) g' = Normal (v'', g'')"
using a1 25
       expr.simps(17) extractValueType.simps calculation by (simp split:if_split_asm option.splits
prod.splits)

    moreover have "rexp i ev cd st g'' = Normal ((v, t), g4')" using a1 25 expr.simps(17)
extractValueType.simps calculation by (simp split:if_split_asm option.splits prod.splits)
    ultimately show "
       (exprResultCon ev cd st v t)"
      using a1 25(1)[of v' g' v'' g''] by auto
  qed
next
  case (26 i xe e cd st g)
  show ?case
  proof (intros)
    fix v t g4' assume a1:"local.expr (CALL i xe) e cd st g = Normal (( v, t), g4') ∧
                           TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd
                           ∧ fullyInitialised e (Accounts st) (Stack st)"
    then obtain v' where a10:"assert Gas ((<) (costs_e (CALL i xe) e cd st)) g = Normal (v', g)" using
expr.simps(18) a1 by (simp split:if_split_asm option.splits prod.splits)
    then obtain v'' g'' where a20:"modify (λg. g - costs_e (CALL i xe) e cd st) g = Normal (v'', g'')"
using expr.simps(18) by (simp split:if_split_asm option.splits prod.splits)
    then obtain vb where a30:"option Err (λ_. ep $$ Contract e) g'' = Normal (vb, g'')" using a1
expr.simps(18) by (simp split:if_split_asm option.splits prod.splits)
    then obtain ct dud where a40:"vb = (ct, dud)" using a1 expr.simps(18) by (simp split:if_split_asm
option.splits prod.splits)
    then obtain vc where a50:"(case ct $$ i of None ⇒ throw Err | Some (Function (fp, True, xa)) ⇒
throw Err | Some
       (Function (fp, False, xa)) ⇒ return (fp, xa) | Some _ ⇒ throw Err) g''
                          = Normal (vc, g'')" using a1 expr.simps(18) a30 by (simp split:if_split_asm
option.splits prod.splits member.splits result.split_asm bool.splits)
    then obtain fp x where a60:"vc = (fp, x)" using a1 expr.simps(18) a30 a40 by (auto
split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits)

    then obtain e' where a70:"e' = ffold_init ct (emptyEnv (Address e) (Contract e) (Sender e) (Svalue
e)) (fmdom ct)"
      using a1 expr.simps(18) by (simp split:if_split_asm option.splits prod.splits)
    then have sameAddSto:"Address e = Address e'" by simp
    then have "Storage st (Address e') = Storage st (Address e)" by simp
    have a72:"(∃c. Type (Accounts st (Address e')) = Some (atype.Contract c) ∧ Contract e' = c)" using
a1 unfolding fullyInitialised_def a70 by simp
    have e'IsStoreloc:"∀x y . (Denvalue e' $$ x = Some y ⟶ snd y = Storeloc x)" using a70 unfolding
ffold_init_def
      using ffoldInitAllLocsStorage[of ct e e e "Svalue e" "fmdom ct"] by simp


  — Load the Denvalue from e to e', excluding storage variables
    obtain ev cda k m g''' where a80:"load False fp xe e' emptyTypedStore emptyStore (Memory st) e cd
st g'' = Normal ((ev, cda, k, m), g''')"
      using expr.simps(18) a1 a10 a20 a30 a40 a50 a60 a70 by (auto split:if_split_asm option.splits
prod.splits member.splits result.split_asm bool.splits)
    then have a85:"(∃c. Type (Accounts st (Address ev)) = Some (atype.Contract c) ∧ Contract ev = c)"
using a72
      using expressions_with_gas.msel_ssel_expr_load_rexp_gas(4) statement_with_gas_axioms
statement_with_gas_def by auto
    have a90:"ep $$ Contract e = Some(ct,dud)" using a30 a40 by (simp split:option.splits)
    have a91:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
      using a1 unfolding fullyInitialised_def by simp
```

```
    have a92:"ep $$ Contract e' = Some(ct,dud)" using a30 a40 a70 by (simp split:option.splits)
    then have a95:" (∀id v'. ct $$ id = Some (Var v') ⟶ ((Denvalue e') $$ id = Some (type.Storage
v', Storeloc id)))" using a70 ffold_init_fmap unfolding ffold_init_def
      by (metis emptyEnv_denvalue fmdom_notD fmdom_notI fmempty_lookup not_Some_eq)
    have sameCont:"Contract e = Contract e' ∧ Address e = Address e'" using a30 a40
      by (simp add: a70)
    have a97:"∀id. id |∈| fmdom (Denvalue e') ⟶ id |∈| fmdom ct" using ffold_init_emptyDen a70 by
auto

    obtain xa ya where a100:"TypeSafe e' (Accounts st) emptyStore (Memory st) (Storage st)
emptyTypedStore
                              ∧ (Denvalue e' $$ xa = Some ya ⟶ snd ya = Storeloc xa)
                              ∧ (Denvalue e' $$ xa = Some ya ⟶ (∃t1. ct $$ xa = Some (Var t1) ∧ fst
ya = type.Storage t1))"
      using ffoldInitTypeSafe[of "Accounts st" "(Storage st) " " (Svalue e)" "Memory st" e ct dud
"Address e" "Sender e"] a70 ffold_init_def a1 a90 a91
      unfolding TypeSafe_def by simp
    then have a103:"TypeSafe e' (Accounts st) emptyStore (Memory st) (Storage st) emptyTypedStore" by
blast

    have a101:" ∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp (Memory st) locs" by simp
    have a102: " Memory st = Memory st" by simp

    have a104:"ncpDenvalueLimit e' e emptyStore (Stack st) (Memory st)" unfolding ncpDenvalueLimit_def
    proof intros
      fix tp' locs p i
      assume "(type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e') ∧ accessStore locs emptyStore =
Some (KMemptr p) ∧ i <
          Toploc (Memory st) ∧ LSubPrefL2 p (ShowL_nat i)"
      then show "∃tp'' loc2 p'.
          (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue e) ∧
          accessStore loc2 (Stack st) = Some (KMemptr p') ∧ (p' = p ∧
              tp'' = tp' ∨ (∃len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr tp' p'
              p))"
        unfolding accessStore_def emptyStore_def by simp
    qed
    have a105:"ncpElementsNoSubPref (Memory st) (Memory st)" using ncpElementsNoSubPref_sameMem a1 by
blast
    have a106:" ncpOMemInDMem (Memory st) (Memory st)"
      unfolding ncpOMemInDMem_def by blast
    have ncp:"ncpNewSelfPoint (Memory st) (Memory st)" unfolding ncpNewSelfPoint_def by auto
    have "(∃c. Type (Accounts st (Address e')) = Some (atype.Contract c) ∧ Contract e' = c)"
      using a72 by simp
    moreover have a96:
      "(∀id v. (ct $$ id = Some (Var v)) = (Denvalue e' $$ id = Some (type.Storage v, Storeloc id)))"
      using a70 unfolding ffold_init_def
      by (metis ffoldInit_var_storage_mapping ffoldInit_var_storage_mapping_rev)
    moreover have "fullyInitialised e' (Accounts st) (Stack st)" unfolding fullyInitialised_def
      using calculation e'IsStoreloc ncp a92 denvalue.distinct(1) fmranE snd_conv
      by (metis denvalue.inject(2))

    moreover have "fullyInitialised e' (Accounts st) emptyStore" unfolding fullyInitialised_def
      using calculation e'IsStoreloc
      using fullyInitialised_def by fastforce
    ultimately have a110:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧ fullyInitialised ev
(Accounts st) k
        ∧ (∀locs tp. MCon tp (state.Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (state.Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e k (Stack st) (state.Memory st) ∧
        ncpOMemInDMem (state.Memory st) m ∧
        ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st) m"
      using "26.hyps"(1)[OF a10 _ a30 a40 a50 a60 a70 ] a20 a80 a90 a101 a102 a100 a103 a104 a105 a106
a1 ncp sameAddSto by auto
```

```
    have a120:"expr x ev cda (st(|Stack := k, Memory := m|)) g''' = Normal ((v, t), g4')" using a1
expr.simps(18) a10 a20 a30
        a40 a50 a60 a70 a80 a90 by (auto split:option.split_asm stackvalue.split_asm result.split_asm
if_split_asm)
    then have a125:"(case v of KValue x ⇒ return (v,t)
      | KCDptr cdloc ⇒ throw Err
      | KMemptr memloc ⇒ return (v, t)
      | KStoptr storloc ⇒ return (v, t)) g4' = Normal ((v, t), g4')" using a1 expr.simps(18) a10 a20
a30 a40 a50 a60 a70 a80 a90
      by (auto split:if_splits option.splits stackvalue.splits member.splits prod.splits result.splits
bool.splits)
    then have a127:"TypeSafe ev (Accounts (st(|Stack := k, Memory := m|)))
                          (Stack (st(|Stack := k, Memory := m|)))
                          (Memory (st(|Stack := k, Memory := m|)))
                          (Storage (st(|Stack := k, Memory := m|))) cda" using a110 by simp
    then have a128:"TypeSafe ev (Accounts st) k m (Storage st) cda" by simp
    have "(∀id ct dud v. ep $$ Contract ev = Some (ct, dud) ∧ ct $$ id = Some (Var v) ⟶ (∃t.
Denvalue
        ev $$ id = Some (type.Storage t, Storeloc id)))" using a80 a95 load_denval_existing_remain
      by (metis a92 fst_eqD msel_ssel_expr_load_rexp_gas(4) option.inject)
    then have "fullyInitialised ev (Accounts st) k"using a110 by simp
    then have a130:"exprResultCon ev cda (st(|Stack := k, Memory := m|)) v t"
      using "26.hyps"(2) a1 a10 a20 a30 a40 a50 a60 a70 a80 a85 a90 a120 a128 by simp

    then show " exprResultCon e cd st v t"
    proof(cases t)
      case (Value x1)
      then show ?thesis using a130 by simp
    next
      case (Calldata x2)

      then show ?thesis using Calldata a1 expr.simps(18)[of i xe e cd st g ] a130 a10 a20 a30 a40 a50
a60 a70
        by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm
bool.splits stackvalue.splits)
    next
      case (Memory x3)
      then have "MCon x3 (state.Memory (st(|Stack := k, Memory := m|))) (extractValueType v)"
        using a130 by simp
      then show ?thesis using Memory a1 expr.simps(18)[of i xe e cd st g ] a130 a10 a20 a30 a40 a50
a60 a70
        by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm
bool.splits stackvalue.splits)
    next
      case (Storage x4)
      have sameAddress:"Address ev = Address e" using a80 a70 by (simp add:msel_ssel_expr_load_rexp_gas(4))

      have ctype_addr_unique:
        "(∀c' adv'. Type (Accounts st adv') = Some (atype.Contract c') ∧ (Address e) = adv' ⟶ c' =
Contract e)"
        using a1 a30 unfolding TypeSafe_def
        using fullyInitialised_def by simp

      have " ep $$ Contract e' = Some (ct, dud)" using a90 a70 by simp
      then have "(∀t l. (type.Storage t, Storeloc l) |∈| fmran (Denvalue e') ⟶ ct $$ l = Some (Var
t))"
        by (metis a96 denvalue.inject(2) e'IsStoreloc fmranE snd_conv)

      have a500:"SCon x4 (extractValueType v) (Storage (st(|Stack := k, Memory := m|)) (Address ev)) ∧
        (∃xx. v = KStoptr xx) ∧
        (∃stloc tp'' .
            (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue ev) ∧
```

```
          (tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4 stloc
(extractValueType v)))"
        using a130 Storage by simp
      then obtain stloc tp'' where def:"(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue ev) ∧
          (tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4 stloc
(extractValueType v))"
        using a110 by auto
      then have a550:"tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp''
x4 stloc (extractValueType v)" using def by simp
      have "(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue ev)" using def by auto
      then have "(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e')"
      proof -
        have fi_ev: "fullyInitialised ev (Accounts st) k" using a110 by simp
        then obtain c_ev ct_ev dud_ev where
          fi_ev1: "Type (Accounts st (Address ev)) = Some (atype.Contract c_ev)"
          and fi_ev2: "Contract ev = c_ev"
          and fi_ev3: "ep $$ c_ev = Some (ct_ev, dud_ev)"
          and fi_ev4: "∀ id v. (ct_ev $$ id = Some (Var v)) = (Denvalue ev $$ id = Some (type.Storage
v, Storeloc id))"
          and fi_ev5: "∀ id v loc. Denvalue ev $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc"
          unfolding fullyInitialised_def by blast
        from ‹(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue ev)›
        obtain id where id_def: "Denvalue ev $$ id = Some (type.Storage tp'', Storeloc stloc)"
          using fmranE
          by fast
        then have id_eq: "id = stloc" using fi_ev5 by blast
        then have den_ev_lookup: "Denvalue ev $$ stloc = Some (type.Storage tp'', Storeloc stloc)"
          using id_def by simp
        have c_ev_eq: "c_ev = Contract e"
          using ctype_addr_unique fi_ev1 sameAddress
          by simp
        then have ct_ev_eq: "ct_ev = ct"
          using fi_ev3 a90 by auto
        then have ct_lookup: "ct $$ stloc = Some (Var tp'')"
          using fi_ev4 den_ev_lookup by blast
        then have den_e'_lookup: "Denvalue e' $$ stloc = Some (type.Storage tp'', Storeloc stloc)"
          using a96 by blast
        then show ?thesis by (simp add: fmranI)
      qed

      then have "(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e)" using a100 a1
          member.inject(3) a90 fmranI unfolding fullyInitialised_def
        by (metis Pair_inject ‹∀ t l. (type.Storage t, Storeloc l) |∈| fmran (Denvalue e') ⟶ ct $$ l
= Some (Var t)›
            option.inject)


      then show ?thesis using Storage a130 def a550 sameAddress by fastforce
    qed
  qed

next
  case (27 ad i xe e cd st g)
  show ?case
  proof (intros)
    fix v t g4' assume a1: "local.expr (ECALL ad i xe) e cd st g = Normal ((v, t), g4') ∧
       TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ fullyInitialised e (Accounts
st) (Stack st)"

    then obtain v' where a10:"assert Gas ((<) (costs_e (ECALL ad i xe) e cd st)) g = Normal (v', g)"
using expr.simps(19) by (simp split:if_split_asm option.splits prod.splits)
    then obtain v'' g'' where a20:"modify (λg. g - costs_e (ECALL ad i xe) e cd st) g = Normal (v'',
g'')" using expr.simps(19) by (simp split:if_split_asm option.splits prod.splits)
```

then obtain *g''' kad* **where** *a30:"expr ad e cd st g'' = Normal (kad, g''')"* **using** *expr.simps(19) a1*
by *(simp split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits)*
then obtain *adv g'''' ''* **where** *a40:"(case kad of (KValue adv, Value TAddr) ⇒ return adv | _ ⇒ throw Err) g''' = Normal (adv, g'''')"* **using** *expr.simps(19) a1 a20*
by *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then have *a50:"adv ≠ Address e"* **using** *expr.simps(19) a1 a20 a30*
by *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then obtain *c ga5* **where** *a60:"(case Type (Accounts st adv) of Some (atype.Contract c) ⇒ return c | _ ⇒ throw Err)*
g'''' = Normal (c, ga5)" **using** *expr.simps(19) a1 a20 a30 a40 a50*
by *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then obtain *vb* **where** *a70:"option Err (λ_. ep $$ c) ga5 = Normal (vb, ga5)"*
using *a1 expr.simps(19) a20 a30 a40 a50* **by** *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then obtain *ct dud* **where** *a80:"vb = (ct, dud)"* **using** *a1 expr.simps(19) a20 a30 a40 a50*
by *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then obtain *fp x* **where** *a90:"(case ct $$ i of Some (Function (fp, True, x)) ⇒ return (fp, x) | _ ⇒ throw Err) ga5*
= Normal ((fp ,x), ga5)" **using** *a1 expr.simps(19) a20 a30 a40 a50 a60 a70*
by *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits type.splits)*
then obtain *e'* **where** *a100:"e' = ffold_init ct (emptyEnv adv c (Address e) (ShowL$_{nat}$ 0)) (fmdom ct)"* **using** *a1*
expr.simps(19) **by** *(simp split:if_split_asm option.splits prod.splits)*
then obtain *ev cda k m ga6* **where** *a110:"load True fp xe e' emptyTypedStore emptyStore emptyTypedStore e cd st ga5 = Normal ((ev, cda, k, m), ga6)"*
using *expr.simps(19) a1 a10 a20 a30 a40 a50 a60 a70 a80 a90* **by** *(auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits)*
have *a120:"ep $$ c = Some(ct,dud)"* **using** *a80 a70* **by** *(simp split:option.splits)*

then obtain *contracte::environment* **where** *a125:"c = (Contract contracte)"*
by *(metis Environment.select_convs(2))*
have *c1:" safeContract (Accounts st) (Storage st)"* **using** *a1* **unfolding** *TypeSafe_def* **by** *simp*
have *c2:"balanceTypes (Accounts st)"* **using** *a1* **unfolding** *TypeSafe_def* **by** *simp*
then have *"typeCon (TUInt b256) (ShowL$_{nat}$ 0)"*
using *checkUInt_def[of b256 "STR ''0''"]* **by** *Solidity_Symbex.solidity_symbex*

then have *c3:"svalueTypes (ShowL$_{nat}$ 0)"* **by** *(simp add: svalueTypes_def)*
have *c15:"AddressTypes (Accounts st)"* **using** *a1* **unfolding** *TypeSafe_def* **by** *simp*
then have *c5:"∀ adv. case Type (Accounts st adv) of None ⇒ True | Some EOA ⇒ True | Some (atype.Contract*
c) ⇒ addressFormat adv ∧ c |∈| fmdom ep"
unfolding *AddressTypes_def* **using** *a125 a70 a120* **by** *blast*

have *c8:"lessThanTopLocs emptyTypedStore"* **using** *typedEmptyTopLocs* **by** *metis*
have *c9:"subPrefixStructuralConsistency emptyTypedStore"*
unfolding *subPrefixStructuralConsistency_def accessTypeStore_def emptyTypedStore_def* **by** *simp*
have *c10:"SomeValSomeTyp emptyTypedStore"* **unfolding** *SomeValSomeTyp_def emptyTypedStore_def accessStore_def accessTypeStore_def* **by** *simp*
have *c11:"addressFormat adv"* **using** *c5 a60 a120* **by** *(simp split:option.splits atype.splits)*
have *c12: "addressFormat (Address e)"* **using** *a1* **unfolding** *TypeSafe_def* **by** *simp*
have *c13:"Type (Accounts st adv) = Some (atype.Contract (Contract contracte))"*
using *a60 a125* **by** *(auto split:option.splits atype.splits)*
have *a128:"TypeSafe e' (Accounts st) emptyStore emptyTypedStore (Storage st) emptyTypedStore"*
using *a100 a125 a120*
unfolding *ffold_init_def*
using *ffoldInitTypeSafe[OF c1 c2 c3 c8 _ c11 c12 c15 c9 c10 c13]* **by** *metis*

have *a129:"(∀ t l p. (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some (KStoptr p) ⟶*

```
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))"
      using c5 c3 a1 a120 a125 a100 unfolding fullyInitialised_def by metis
    obtain xa ya where "(Denvalue e' $$ xa = Some ya ⟶ snd ya = Storeloc xa)
        ∧ (Denvalue e' $$ xa = Some ya ⟶ (∃t1. ct $$ xa = Some (Var t1) ∧ fst ya = type.Storage
t1))"
      using a100 by (metis(full_types) denvalue.distinct(1) option.inject snd_conv)
    then have a130:"TypeSafe e' (Accounts st) emptyStore emptyTypedStore (Storage st) emptyTypedStore
∧ (∀t l p. (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))
∧ (Denvalue e' $$ xa = Some ya ⟶ snd ya = Storeloc xa)
        ∧ (Denvalue e' $$ xa = Some ya ⟶ (∃t1. ct $$ xa = Some (Var t1) ∧ fst ya = type.Storage
t1))"
      using a128 a129 by simp
    have ttt:"(∃c ct dud.
      Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
      environment.Contract e = c ∧
      ep $$ c = Some (ct, dud) ∧
      (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc id))) ∧
      (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
      (∀t l p.
        (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)))"
      using a1 unfolding fullyInitialised_def
      using c5 by force

    have "Contract ev = Contract e'" using a110
      by (simp add: msel_ssel_expr_load_rexp_gas(4))
    then have "Contract ev = c" using a100 by simp

    have h1:"(∃c. Type (Accounts (st(|Stack := k, Memory := m|)) (Address ev)) = Some (atype.Contract c)
∧ Contract ev = c)"
    proof -
      have load_same: "Address ev = Address e' ∧ Contract ev = Contract e'"
        using a110 by (simp add: msel_ssel_expr_load_rexp_gas(4))
      have type_adv: "Type (Accounts st adv) = Some (atype.Contract c)"
        using a60 by (auto split: option.splits atype.splits)
      have "Type (Accounts (st(|Stack := k, Memory := m|)) (Address ev)) = Some (atype.Contract c) ∧
Contract ev = c"
        using load_same type_adv a100 by simp
      then show ?thesis by blast
    qed
    then have a95:" (∀id v'. ct $$ id = Some (Var v') ⟶ ((Denvalue e') $$ id = Some (type.Storage
v', Storeloc id)))"
      using a100 ffold_init_fmap unfolding ffold_init_def
      by (simp add: ffold_init_fmdom)
    then have h2:"(∀id ct dud v. ep $$ Contract ev = Some (ct, dud) ∧ ct $$ id = Some (Var v) ⟶
                  (∃t. Denvalue ev $$ id = Some (type.Storage t, Storeloc id)))"
      using a100 a110 a90 load_denval_existing_remain(4)[of True fp xe e' emptyTypedStore emptyStore
emptyTypedStore e cd st ga5 ev cda k m ga6]
      by (metis ‹Contract ev = c› a120 option.inject prod.inject)


    have adde':"Address e' = adv"
      by (simp add: a100)
    have e'IsC:"Contract e' = c" using a100 by simp
    have "Type (Accounts st (Address e')) = Some (atype.Contract c)" using adde' a60 by (simp
split:option.splits atype.splits)
    then have "(∃c. Type (Accounts st (Address e')) = Some (atype.Contract c) ∧ Contract e' = c)"
using adde' e'IsC by blast
    moreover have "(∀id ct dud v. ep $$ Contract e' = Some (ct, dud) ∧ ct $$ id = Some (Var v) ⟶
(Denvalue e' $$ id = Some (type.Storage v, Storeloc id)))"
      using ‹Contract ev = c› ‹Contract ev = Contract e'› a120 a95 by simp
```

```
    moreover have "(∀ id v loc. Denvalue e' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
    proof (intros)
      fix id v loc
      assume h: "Denvalue e' $$ id = Some (type.Storage v, Storeloc loc)"
      have "(Denvalue e' $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v, Storeloc
loc) = Storeloc id)"
        using a100 a125 unfolding ffold_init_def
        using ffoldInitAllLocsStorage[
          of ct "emptyEnv adv c (Address e) (ShowL_nat 0)" contracte
             "emptyEnv adv c (Address e) (ShowL_nat 0)" "ShowL_nat 0" "fmdom ct"
        ]
        by simp
      then show "id = loc" using h by simp
    qed


    moreover have "fullyInitialised e' (Accounts st) emptyStore"
      using calculation(1,3) ‹Contract ev = c› ‹Contract ev = Contract e'› a120 a95 a100 c5
ffold_init_emptyDen_ran ttt
      unfolding fullyInitialised_def unfolding accessStore_def emptyStore_def using ffold_init_def
fmdomI fmdom_notI fmempty_lookup store.select_convs(1)
      by metis


    ultimately have a140:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧ fullyInitialised ev
(Accounts st) k"
      using "27.hyps"(2)[OF a10 _ a30 a40 _ a60 a70 a80 a90 _ a100, of v'' _ fp ] a1 a20 a50 a110 a120
a130 by auto

    have a150:"expr x ev cda (st(|Stack := k, Memory := m|)) ga6 = Normal ((v, t), g4')"
      using a1 expr.simps(19)[of ad i xe e cd st g ] a10 a20 a30 a40 a50 a60 a70 a80 a90 a100 a110 a120
a130
      by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm bool.splits
atype.splits type.splits stackvalue.splits)
    then have a160:"(case v of KValue x ⇒ return (v,t)
      | KCDptr cdloc ⇒ throw Err
      | KMemptr memloc ⇒ throw Err
      | KStoptr storloc ⇒ throw Err) g4' = Normal ((v, t), g4')" using a1 expr.simps(19) a10 a20 a30
a40 a50 a60 a70 a80 a90
      by (auto split:if_splits option.splits stackvalue.splits member.splits prod.splits result.splits
bool.splits)

    then have "TypeSafe ev (Accounts (st(|Stack := k, Memory := m|)))
                           (Stack (st(|Stack := k, Memory := m|)))
                           (Memory (st(|Stack := k, Memory := m|)))
                           (Storage (st(|Stack := k, Memory := m|))) cda" using a160 a140 by simp

    then have a170:"exprResultCon ev cda (st(|Stack := k, Memory := m|)) v t"
      using "27.hyps"(3)[of v' g v'' g'' kad g''' adv g'''' _ _ c ga5 vb ga5 ct dud "(fp,x)" ga5 fp x
e' "(ev, cda, k, m)" ga6 ev "(cda, k, m)" cda "(k,m)" k m ]
        a1 a10 a20 a30 a40 a50 a60 a70 a80 a90 a100 a110 a120 a130 a140 a150 a160 h1 h2 by simp

    then show "exprResultCon e cd st v t"
    proof(cases t)
      case (Value x1)
      then show ?thesis using a170 by simp
    next
      case (Calldata x2)
      then show ?thesis using Calldata a1 expr.simps(19)[of ad i xe e cd st g ] a10 a20 a30 a40 a50
a60 a70 a80 a90 a100 a110 a120 a130 a140 a150 a160 a170
        by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm
bool.splits atype.splits type.splits stackvalue.splits)
    next
      case (Memory x3)
```

```
      then show ?thesis using Memory a1 expr.simps(18)[of i xe e cd st g ]a10 a20 a30 a40 a50 a60 a70
a80 a90 a100 a110 a120 a130 a140 a150 a160 a170
        by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm
bool.splits atype.splits type.splits stackvalue.splits)
    next
      case (Storage x4)
      then show ?thesis using a1 expr.simps(18)[of i xe e cd st g ]a10 a20 a30 a40 a50 a60 a70 a80 a90
a100 a110 a120 a130 a140 a150 a160 a170
        by (auto split:if_split_asm option.splits prod.splits member.splits result.split_asm
bool.splits atype.splits type.splits stackvalue.splits)
    qed

  qed

next
  case (28 cp i_p t_p pl ex el e_v' cd' sck' mem' e_v cd st g)
  show ?case
  proof(intros)
    fix ev cda k m g' loc t ls tp locs id' v''
    assume "load cp ((i_p, t_p) # pl) (ex # el) e_v' cd' sck' mem' e_v cd st g = Normal ((ev, cda, k, m),
g') ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs) ∧
        Toploc (Memory st) ≤ Toploc mem' ∧
        ncpDenvalueLimit e_v' e_v sck' (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) mem' ∧ ncpElementsNoSubPref (Memory st) mem' ∧ ncpNewSelfPoint
(Memory st) mem' ∧ Address e_v = Address e_v'
        ) ∧
       TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧
       TypeSafe e_v' (Accounts st) sck' mem' (Storage st) cd' ∧
       fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e_v' (Accounts st) sck'"
    then have as1: "local.load cp ((i_p, t_p) # pl) (ex # el) e_v' cd' sck' mem' e_v cd st g = Normal
((ev, cda, k, m), g')"
      and as2:"(¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs) ∧ Toploc (Memory st) ≤ Toploc mem'
∧
        ncpDenvalueLimit e_v' e_v sck' (Stack st) (Memory st) ∧ ncpOMemInDMem (Memory st) mem'
        ∧ ncpElementsNoSubPref (Memory st) mem'
        ∧ ncpNewSelfPoint (Memory st) mem' ∧ Address e_v = Address e_v')"
      and as6:"TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ TypeSafe e_v'
(Accounts st) sck' mem' (Storage st) cd'"
      and as7:"fullyInitialised e_v (Accounts st) (Stack st)"
      and as8:"fullyInitialised e_v' (Accounts st) sck'"by blast+
    then have a2:"TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd" by blast
    then obtain x g4' where a7:"expr ex e_v cd st g = Normal (x, g4')"
      using as1 load.simps(1)[of cp i_p t_p pl ex el e_v' cd' sck' mem' e_v cd st g]
      by (simp split:if_splits type.splits result.splits prod.splits option.splits)
    then have a8:"∃ v t. expr ex e_v cd st g = Normal ((v, t), g4')" using as1 by simp
    then obtain v t where a10:"expr ex e_v cd st g = Normal ((v, t), g4')"
      using as1 load.simps(1)[of cp i_p t_p pl ex el e_v' cd' sck' mem' e_v cd st g]
      by (auto split: result.splits prod.splits option.splits)
    then obtain c m' k' e where a15:"decl i_p t_p (Some (v,t)) cp cd (Memory st) (Storage st (Address
e_v)) (cd', mem', sck', e_v') = Some (c, m', k', e)"
      using as1 load.simps(1) by (auto split:if_splits type.splits result.splits prod.splits
option.splits)
    have sameAddDecl:"Address e_v' = Address e" using decl_env a15 by simp

    have a18: "(if ¬cp then (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs) else True)"
using as1 as2 by simp
    then have a20:"load cp pl el e c k' m' e_v cd st g4' = Normal ((ev, cda, k, m), g')"
      using as1 load.simps(1)[of cp i_p t_p pl ex el e_v' cd' sck' mem' e_v cd st g] a10 a15
      by (simp split:type.splits if_splits option.splits)

    have adds:"Address e = Address e_v'" using a15
```

```
      by (simp add: decl_env)
    moreover have sameCon:"Contract e = Contract e_v'" using a15
      using decl_env by auto


    have none:"Denvalue e_v' $$ i_p = None" using as1 unfolding load.simps by (auto split:option.splits
type.splits)

    then have cc0:"TypeSafe ev (Accounts st) k m (Storage st) cda
                ∧ (¬ cp ⟶ (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
                           Toploc (Memory st) ≤ Toploc m ∧ ncpElementsNoSubPref (Memory st) m
∧ ncpOMemInDMem (Memory st) m
∧ ncpNewSelfPoint (Memory st) m ∧ ncpDenvalueLimit ev e_v k (Stack st) (Memory st))
∧ fullyInitialised ev (Accounts st) k
"
    proof(cases t_p)
      case (Value x1)
      have fiE:"fullyInitialised e (Accounts st) k'"
      proof -
        obtain ct dud c'' where c''def:"
             Type (Accounts st (Address e_v')) = Some (atype.Contract c'') ∧
             environment.Contract e_v' = c'' ∧
             ep $$ c'' = Some (ct, dud) ∧
             (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v' $$ id = Some (type.Storage v, Storeloc
id))) ∧
             (∀ id v loc. Denvalue e_v' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
             (∀ t l p.
                 (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v') ∧ accessStore l sck' = Some
(KStoptr p) ⟶
                 (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v') ∧ CompStoType t' t
l' p))"
          using sameAddDecl as8 sameCon unfolding fullyInitialised_def
          by presburger
        have "Type (Accounts st (Address e)) = Some (atype.Contract c'') ∧ environment.Contract e = c''
∧
           ep $$ c'' = Some (ct, dud)" using c''def adds
          by (metis adds c''def sameCon)
        moreover have "(∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v,
Storeloc id)))"
          using c''def a15 as8 decl_env_monotonic decl_env_not_i decl_env_storlocs_unchanged
          by fast
        moreover have "(∀ t l p. (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l
k' = Some (KStoptr p)
                        ⟶ (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧
CompStoType t' t l' p))"
        proof intros
          fix t'' l p
          assume cc1:"(type.Storage t'', Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KStoptr p)"
          then obtain ii where iidef:"(Denvalue e) $$ ii = Some (type.Storage t'', Stackloc l)" by
blast

          then show "∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t'
t'' l' p"
          proof(cases "ii = i_p")
            case True
            then show ?thesis
              by (metis Value a15 decl_env_storage iidef none type.distinct(5))
          next
            case False
            then have "(Denvalue e_v') $$ ii = Some (type.Storage t'', Stackloc l)"
              using a15 as8 decl_env_monotonic[OF a15] decl_env_not_i[OF a15]
decl_env_storlocs_unchanged[OF a15] iidef by metis
            moreover have acc:"accessStore l sck' = accessStore l k'" using Value a15
```

```
decl_StorageStack_options[OF a15 _ iidef ] cc1 by simp
          ultimately obtain t2 l2 where t2Def:"((type.Storage t2, Storeloc l2) |∈| fmran (Denvalue
e_v') ∧ CompStoType t2 t'' l2 p)" using cc1 c''def fmranI by metis
          then have "(type.Storage t2, Storeloc l2) |∈| fmran (Denvalue e)" using
decl_env_monotonic[OF a15] fmranI by fast
          then show ?thesis using t2Def by metis
        qed
      qed
      moreover have "(∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc)"
        by (metis (no_types, lifting) a15 c''def decl_env_not_i decl_env_storlocs_unchanged)
      ultimately show "fullyInitialised e (Accounts st) k'" unfolding fullyInitialised_def by blast
    qed


    then have a22:"exprResultCon e_v cd st v t"
      using as6 a10 "28.hyps"(1) a18 Value none as7 as8 by simp
    then have a25:" valueCon e_v cd st v t" by (simp split:type.splits)
    have a26:"∀struct.
     t = type.Memory struct ⟶
     (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧ accessStore stloc
(Stack st) = Some (KMemptr p) ∧ (tp'' = struct ∧ v = (KMemptr p)
    ∨ (∃len arr. (extractValueType v) ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len
arr struct p (extractValueType v))))" using a22 by auto
    have a27:"∀struct. t = Calldata struct ⟶

    (∃stloc tp'' p.
        (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
        accessStore stloc (Stack st) = Some (KCDptr p) ∧
        (tp'' = struct ∧ v = KCDptr p ∨ (∃len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr
∧
            CompMemType cd len arr struct p (extractValueType v))))" using a22 by auto
    have a28:" ∀struct.
     t = type.Storage struct ⟶
     (∃stloc tp'' .
         (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
         (tp'' = struct ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' struct
stloc (extractValueType v)))" using a22 by auto
    have sameAddDecl:"Address e_v' = Address e" using decl_env a15 by simp
    have a30:"TypeSafe e (Accounts st) k' m' (Storage st) c ∧
  (¬ cp ⟶
   (∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
   Toploc (Memory st) ≤ Toploc m' ∧
   ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
   ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st)
m') ∧
  Toploc mem' ≤ Toploc m' "
      using typeSafeDecl[of e_v st cd v t e_v' sck' mem' cd' i_p t_p cp c m' k' e] a2 as1 as2 a15 a25
Value a18 a26 as6 a27 a28 fiE by blast
    have notSto:"¬ (case t_p of type.Storage x ⇒ cp | _ ⇒ False)"
      using Value by simp
    have cc1:"∀ev cda k m g'.
     local.load cp pl el e c k' m' e_v cd st g4' = Normal ((ev, cda, k, m), g') ∧
     (¬ cp ⟶
      (∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
      Toploc (Memory st) ≤ Toploc m' ∧
      ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
      ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory
st) m'∧ Address e_v = Address e) ∧
     TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧
     TypeSafe e (Accounts st) k' m' (Storage st) c ∧ fullyInitialised e_v (Accounts st) (Stack st) ∧
fullyInitialised e (Accounts st) k' ⟶
     TypeSafe ev (Accounts st) k m (Storage st) cda ∧
    fullyInitialised ev (Accounts st) k ∧
```

```
      (¬ cp ⟶
       (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
       Toploc (Memory st) ≤ Toploc m ∧
       ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
       ncpOMemInDMem (Memory st) m ∧ ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st)
m)"
       using "28.hyps"(2)[OF notSto none a10 _, of v t _ _ c _ m' _ k' e] as1 a15 Value a2 as2 as6 as7
as8 by force
     then have cc2:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧
      fullyInitialised ev (Accounts st) k ∧
      (¬ cp ⟶
       (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
       Toploc (Memory st) ≤ Toploc m ∧
       ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
       ncpOMemInDMem (Memory st) m ∧ ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st)
m)"
       using a20 a2 as1 as2 a30 as7 as8 fiE sameAddDecl by simp

     moreover have "TypeSafe ev (Accounts st) k m (Storage st ) cda" using cc2 by simp

     ultimately show " TypeSafe ev (Accounts st) k m (Storage st) cda ∧
         (¬ cp ⟶
          (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
          Toploc (Memory st) ≤ Toploc m ∧
          ncpElementsNoSubPref (Memory st) m ∧
          ncpOMemInDMem (Memory st) m ∧ ncpNewSelfPoint (Memory st) m ∧ ncpDenvalueLimit ev e_v k
(Stack st) (Memory st)) ∧
         fullyInitialised ev (Accounts st) k"
       using fmsubset_alt_def cc2 fmsubset_pred by meson
   next
     case (Calldata x2)
     have fiE:"fullyInitialised e (Accounts st) k'" proof -
       obtain ct dud c'' where c''def:"
            Type (Accounts st (Address e_v')) = Some (atype.Contract c'') ∧
            environment.Contract e_v' = c'' ∧
            ep $$ c'' = Some (ct, dud) ∧
            (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v' $$ id = Some (type.Storage v, Storeloc
id))) ∧
            (∀ id v loc. Denvalue e_v' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
            (∀ t l p.
                (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v') ∧ accessStore l sck' = Some
(KStoptr p) ⟶
                (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v') ∧ CompStoType t' t
l' p))"
         using sameAddDecl as8 sameCon unfolding fullyInitialised_def
         by presburger
       have "Type (Accounts st (Address e)) = Some (atype.Contract c'') ∧ environment.Contract e = c''
∧
          ep $$ c'' = Some (ct, dud)" using c''def adds
         by (metis adds c''def sameCon)
       moreover have "(∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v,
Storeloc id)))"
         using c''def a15 as8 decl_env_monotonic decl_env_not_i decl_env_storlocs_unchanged
         by fast
       moreover have "(∀ t l p. (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l
k' = Some (KStoptr p)
                       ⟶ (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧
CompStoType t' t l' p))"
       proof intros
         fix t'' l p
         assume cc1:"(type.Storage t'', Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KStoptr p)"
         then obtain ii where iidef:"(Denvalue e) $$ ii = Some (type.Storage t'', Stackloc l)" by
blast
```

```
          then show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t'
t'' l' p"
          proof(cases "ii = i_p")
            case True
            then show ?thesis using a15 decl_env_storage iidef none type.distinct(7) Calldata by
fastforce
          next
            case False
            then have "(Denvalue e_v') $$ ii = Some (type.Storage t'', Stackloc l)"
              using a15 as8 decl_env_monotonic[OF a15] decl_env_not_i[OF a15]
decl_env_storlocs_unchanged[OF a15] iidef by metis
            moreover have acc:"accessStore l sck' = accessStore l k'" using Calldata a15
decl_StorageStack_options[OF a15 _ iidef ] cc1 by simp
            ultimately obtain t2 l2 where t2Def:"((type.Storage t2, Storeloc l2) |∈| fmran (Denvalue
e_v') ∧ CompStoType t2 t'' l2 p)" using cc1 c''def fmranI by metis
            then have "(type.Storage t2, Storeloc l2) |∈| fmran (Denvalue e)" using
decl_env_monotonic[OF a15] fmranI by fast
            then show ?thesis using t2Def by metis
          qed
        qed
        moreover have "(∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc)"
          by (metis (no_types, lifting) a15 c''def decl_env_not_i decl_env_storlocs_unchanged)
        ultimately show "fullyInitialised e (Accounts st) k'" unfolding fullyInitialised_def by blast
      qed
      then have a22:"exprResultCon e_v cd st v t"
        using as6 a10 "28.hyps"(1) a18 Calldata as7 as8 none by simp
      then have a25:" valueCon e_v cd st v t" by (simp split:type.splits)
      have a26:"∀struct.
       t = type.Memory struct ⟶
       (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧ accessStore stloc
(Stack st) = Some (KMemptr p) ∧ (tp'' = struct ∧ v = (KMemptr p)
     ∨ (∃len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr
struct p (extractValueType v))))" using a22 by auto
      have a27:"∀struct. t = Calldata struct ⟶

      (∃stloc tp'' p.
          (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
          accessStore stloc (Stack st) = Some (KCDptr p) ∧
          (tp'' = struct ∧ v = KCDptr p ∨ (∃len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr
∧
              CompMemType cd len arr struct p (extractValueType v))))" using a22 by auto

      have a28:" ∀struct.
       t = type.Storage struct ⟶
       (∃stloc tp''.
           (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
           (tp'' = struct ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' struct
stloc (extractValueType v)))" using a22 by auto
      have a30:"TypeSafe e (Accounts st) k' m' (Storage st) c ∧
    (¬ cp ⟶
     (∀locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
     Toploc (Memory st) ≤ Toploc m' ∧
     ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
     ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st)
m') ∧
    Toploc mem' ≤ Toploc m'"
        using typeSafeDecl[of e_v st cd v t e_v' sck' mem' cd' i_p t_p cp c m' k' e] a2 as1 as2 a15 a25
Calldata a18 a26 as6 a27 a28 by blast
      have notSto:"¬ (case t_p of type.Storage x ⇒ cp | _ ⇒ False)"
        using Calldata by simp

      have cc1:"∀ev cda k m g'.
```

```
       local.load cp pl el e c k' m' e_v cd st g4' = Normal ((ev, cda, k, m), g') ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
        Toploc (Memory st) ≤ Toploc m' ∧ ncpDenvalueLimit e e_v k' (Stack st) (Memory st)
        ∧ ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory
st) m' ∧ Address e_v = Address e) ∧
       TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ TypeSafe e (Accounts st) k'
m' (Storage st) c ∧
        fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e (Accounts st) k'
⟶
       TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       fullyInitialised ev (Accounts st) k
∧ (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧
        ncpElementsNoSubPref (Memory st) m ∧
        ncpNewSelfPoint (Memory st) m)"
        using "28.hyps"(2)[OF notSto none a10 _, of v t ] a15 Calldata a2 as1 as2 as6 as7 as8 by force
      then have cc2:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧ ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st)
m)
∧
       fullyInitialised ev (Accounts st) k"
        using a20 a2 as1 a30 as7 as8 fiE sameAddDecl as2 by simp
      moreover have "TypeSafe ev (Accounts st) k m (Storage st ) cda" using a20 a2 as1 a30 cc1
        using cc2 by blast
      ultimately show "TypeSafe ev (Accounts st) k m (Storage st ) cda
∧ (¬ cp ⟶
          (∀ locs tp.
              MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
          Toploc (Memory st) ≤ Toploc m ∧
          ncpElementsNoSubPref (Memory st) m ∧ ncpOMemInDMem (Memory st) m ∧
          ncpNewSelfPoint (Memory st) m ∧
          ncpDenvalueLimit ev e_v k (Stack st)(Memory st))∧
         fullyInitialised ev (Accounts st) k"
        using fmsubset_alt_def cc2 fmsubset_pred
        by meson

    next
      case (Memory x3)
      have fiE:"fullyInitialised e (Accounts st) k'"
      proof -
        obtain ct dud c'' where c''def:"
             Type (Accounts st (Address e_v')) = Some (atype.Contract c'') ∧
             environment.Contract e_v' = c'' ∧
             ep $$ c'' = Some (ct, dud) ∧
             (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v' $$ id = Some (type.Storage v, Storeloc
id))) ∧
              (∀ id v loc. Denvalue e_v' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
             (∀ t l p.
                 (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v') ∧ accessStore l sck' = Some
(KStoptr p) ⟶
                 (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v') ∧ CompStoType t' t
l' p))"
          using sameAddDecl as8 sameCon unfolding fullyInitialised_def
          by presburger
        have "Type (Accounts st (Address e)) = Some (atype.Contract c'') ∧ environment.Contract e = c''
∧
```

```
          ep $$ c'' = Some (ct, dud)" using c''def adds
          by (metis adds c''def sameCon)
        moreover have "(∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v,
Storeloc id)))"
          using c''def a15 as8 decl_env_monotonic decl_env_not_i decl_env_storlocs_unchanged
          by fast
        moreover have "(∀t l p. (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l
k' = Some (KStoptr p)
                        ⟶ (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧
CompStoType t' t l' p))"
        proof intros
          fix t'' l p
          assume cc1:"(type.Storage t'', Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KStoptr p)"
          then obtain ii where iidef:"(Denvalue e) $$ ii = Some (type.Storage t'', Stackloc l)" by
blast

          then show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t'
t'' l' p"
          proof(cases "ii = i_p")
            case True
            then show ?thesis using Memory a15 decl_env_storage iidef none type.distinct(11)
              by (metis)
          next
            case False
            then have "(Denvalue e_v') $$ ii = Some (type.Storage t'', Stackloc l)"
              using a15 as8 decl_env_monotonic[OF a15] decl_env_not_i[OF a15]
decl_env_storlocs_unchanged[OF a15] iidef by metis
            moreover have acc:"accessStore l sck' = accessStore l k'" using Memory a15
decl_StorageStack_options[OF a15 _ iidef ] cc1 by simp
            ultimately obtain t2 l2 where t2Def:"((type.Storage t2, Storeloc l2) |∈| fmran (Denvalue
e_v') ∧ CompStoType t2 t'' l2 p)" using cc1 c''def fmranI by metis
            then have "(type.Storage t2, Storeloc l2) |∈| fmran (Denvalue e)" using
decl_env_monotonic[OF a15] fmranI by fast
            then show ?thesis using t2Def by metis
          qed
        qed
        moreover have "(∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc)"
          by (metis (no_types, lifting) a15 c''def decl_env_not_i decl_env_storlocs_unchanged)
        ultimately show "fullyInitialised e (Accounts st) k'" unfolding fullyInitialised_def by blast
      qed
      then have a22:"exprResultCon e_v cd st v t"
        using as6 a10 "28.hyps"(1) a18 Memory as7 as8 none by simp
      then have a25:" valueCon e_v cd st v t" by (simp split:type.splits)
      have a26:"∀struct.
       t = type.Memory struct ⟶
       (∃stloc tp'' p. (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧ accessStore stloc
(Stack st) = Some (KMemptr p) ∧ (tp'' = struct ∧ v = (KMemptr p)
      ∨ (∃len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr
struct p (extractValueType v))))" using a22 by auto
      have a27:"∀struct. t = Calldata struct ⟶
      (∃stloc tp'' p.
          (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
          accessStore stloc (Stack st) = Some (KCDptr p) ∧
          (tp'' = struct ∧ v = KCDptr p ∨ (∃len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr
∧
              CompMemType cd len arr struct p (extractValueType v))))" using a22 by auto
      have a28:" ∀struct.
       t = type.Storage struct ⟶
       (∃stloc tp''.
          (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
          (tp'' = struct ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' struct
stloc (extractValueType v)))"
```

```
        using a22 by auto
      have a30:" TypeSafe e (Accounts st) k' m' (Storage st) c ∧
    (¬ cp ⟶
     (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
     Toploc (Memory st) ≤ Toploc m' ∧
     ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
     ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st)
m') ∧
    Toploc mem' ≤ Toploc m'"
        using typeSafeDecl[of e_v st cd v t e_v' sck' mem' cd' i_p t_p cp c m' k' e] a2 as1 as2 a15 a25
Memory a18 a26 as6 a27 a28 by blast

      have notSto:"¬ (case t_p of type.Storage x ⇒ cp | _ ⇒ False)"
        using Memory by simp
      have cc1:"∀ ev cda k m g'.
       local.load cp pl el e c k' m' e_v cd st g4' = Normal ((ev, cda, k, m), g') ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
        Toploc (Memory st) ≤ Toploc m' ∧ ncpDenvalueLimit e e_v k' (Stack st) (Memory st)
∧ ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st) m'∧
Address e_v = Address e) ∧
       TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ TypeSafe e (Accounts st) k'
m' (Storage st) c
∧ fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e (Accounts st) k' ⟶
       TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       fullyInitialised ev (Accounts st) k
∧ (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧
        ncpElementsNoSubPref (Memory st) m ∧
        ncpNewSelfPoint (Memory st) m)"
        using "28.hyps"(2)[OF notSto none a10 _, of v t] a15 Memory a2 as1 as2 as6 as7 as8 by force
      then have cc2:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧ ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st)
m)
∧ fullyInitialised ev (Accounts st) k "
        using a20 a2 as1 a30 as7 as8 fiE sameAddDecl as2 by simp
      moreover have "TypeSafe ev (Accounts st) k m (Storage st ) cda" using a20 a2 as1 a30 cc1 cc2 by
blast

      ultimately show "TypeSafe ev (Accounts st) k m (Storage st ) cda
∧ (¬ cp ⟶
           (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
           Toploc (Memory st) ≤ Toploc m ∧
           ncpElementsNoSubPref (Memory st) m ∧ ncpOMemInDMem (Memory st) m ∧
           ncpNewSelfPoint (Memory st) m ∧
           ncpDenvalueLimit ev e_v k (Stack st) (Memory st))
∧ fullyInitialised ev (Accounts st) k"
        using fmsubset_alt_def cc2 fmsubset_pred by meson
    next
      case (Storage x4)
      then have ncp:"¬cp" using as1 load.simps by (auto split:if_splits)
      have a22:"exprResultCon e_v cd st v t"
        using as6 a10 "28.hyps"(1) a18 as7 as8 none ncp Storage by simp
      then have a25:" valueCon e_v cd st v t" by (simp split:type.splits)
      have a26:" ∀ struct.
       t = Calldata struct ⟶
       (∃ stloc tp'' p.
```

```
          (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
          accessStore stloc (Stack st) = Some (KCDptr p) ∧
          (tp'' = struct ∧ v = KCDptr p ∨ (∃ len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr
∧
              CompMemType cd len arr struct p (extractValueType v))))" using a22 by auto
      have a27:" ∀ struct.
       t = type.Memory struct ⟶
       (∃ stloc tp'' p.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
          accessStore stloc (Stack st) = Some (KMemptr p) ∧
          (tp'' = struct ∧ v = KMemptr p ∨ (∃ len arr. extractValueType v ≠ p ∧ tp'' = MTArray len arr
∧
              CompMemType (Memory st) len arr struct p (extractValueType v))))" using a22 by auto
      have a28:" ∀ struct.
       t = type.Storage struct ⟶
       (∃ stloc tp'' .
           (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
           (tp'' = struct ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' struct
stloc (extractValueType v)))" using a22 by auto
      have tEq:"t = type.Storage x4" using Storage a15 decl_storage_tp_params
        by (metis decl_Calldata_tp_match decl_KValue_tp_match decl_Memory_tp_match
type.distinct(11,5,9) type.exhaust)
      have fiE:"fullyInitialised e (Accounts st) k'"
      proof -
        obtain ct dud c'' where c''def:"
             Type (Accounts st (Address e_v')) = Some (atype.Contract c'') ∧
             environment.Contract e_v' = c'' ∧
             ep $$ c'' = Some (ct, dud) ∧
             (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v' $$ id = Some (type.Storage v, Storeloc
id))) ∧
             (∀ id v loc. Denvalue e_v' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
             (∀ t l p.
                 (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v') ∧ accessStore l sck' = Some
(KStoptr p) ⟶
                 (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v') ∧ CompStoType t' t
l' p))"
          using sameAddDecl as8 sameCon unfolding fullyInitialised_def
          by presburger
        have "Type (Accounts st (Address e)) = Some (atype.Contract c'') ∧ environment.Contract e = c''
∧
           ep $$ c'' = Some (ct, dud)" using c''def adds
          by (metis adds c''def sameCon)
        moreover have "(∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v,
Storeloc id)))"
          using c''def a15 as8 decl_env_monotonic decl_env_not_i decl_env_storlocs_unchanged
          by fast
        moreover have "(∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc)"
          by (metis (no_types, lifting) a15 c''def decl_env_not_i decl_env_storlocs_unchanged)
        moreover have "(∀ t'' l p. (type.Storage t'', Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore
l k' = Some (KStoptr p)
                        ⟶ (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧
CompStoType t' t'' l' p))"
        proof intros
          fix t'' l p
          assume cc1:"(type.Storage t'', Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l k' = Some
(KStoptr p)"
          then obtain ii where iidef:"(Denvalue e) $$ ii = Some (type.Storage t'', Stackloc l)" by
blast
          then show "∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t'
t'' l' p"
          proof(cases "ii = i_p")
            case True
            then have iipEq:"Denvalue e $$ i_p = Some (type.Storage t'', Stackloc l)" using iidef by
```

```
simp
            obtain stloc' tp''' where stdef_fi:"
                 (type.Storage tp''', Storeloc stloc') |∈| fmran (Denvalue e_v) ∧
                 (tp''' = x4 ∧ v = KStoptr stloc' ∨ extractValueType v ≠ stloc' ∧ CompStoType tp''' x4
stloc' (extractValueType v))"
              using a28 tEq by blast
            then have "(type.Storage tp''', Storeloc stloc') |∈| fmran (Denvalue e_v')"
            proof -
              obtain c_ev ct_ev dud_ev where ev_fi:"
                   Type (Accounts st (Address e_v)) = Some (atype.Contract c_ev) ∧
                   environment.Contract e_v = c_ev ∧
                   ep $$ c_ev = Some (ct_ev, dud_ev) ∧
                   (∀ id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
                   (∀ id vv. (ct_ev $$ id = Some (Var vv)) = (Denvalue e_v $$ id = Some (type.Storage vv,
Storeloc id)))"
                using as7 unfolding fullyInitialised_def by blast
              obtain c_ev' ct_ev' dud_ev' where ev'_fi:"
                   Type (Accounts st (Address e_v')) = Some (atype.Contract c_ev') ∧
                   environment.Contract e_v' = c_ev' ∧
                   ep $$ c_ev' = Some (ct_ev', dud_ev') ∧
                   (∀ id vv. (ct_ev' $$ id = Some (Var vv)) = (Denvalue e_v' $$ id = Some (type.Storage
vv, Storeloc id)))"
                using as8 unfolding fullyInitialised_def by blast
              have sameADD':"Address e_v = Address e_v'" using as2 sameAddDecl ncp by blast
              have eq_c:"c_ev = c_ev'" using ev_fi ev'_fi sameADD' by simp
              hence eq_ct:"ct_ev = ct_ev'" using ev_fi ev'_fi eq_c by (metis option.inject
prod.inject)
              from stdef_fi obtain k_ev where k_def:"Denvalue e_v $$ k_ev = Some (type.Storage tp''',
Storeloc stloc')"
                using fmlookup_ran_iff by blast
              hence "ct_ev $$ stloc' = Some (Var tp''')"
                using ev_fi by auto
              hence "Denvalue e_v' $$ stloc' = Some (type.Storage tp''', Storeloc stloc')"
                using ev'_fi eq_ct by meson
              thus ?thesis using fmranI by meson
            qed
            then have inE:"(type.Storage tp''', Storeloc stloc') |∈| fmran (Denvalue e)"
              using decl_env_monotonic[OF a15] fmranI by fast
            have pVal:"p = extractValueType v"
            proof -
              have lEq:"l = ShowL_nat (Toploc sck')"
              proof -
                obtain tt where topDen:"Denvalue e $$ i_p = Some (tt, Stackloc (ShowL_nat (Toploc
sck')))"
                  using decl_stack_top[OF a15 none] by blast
                show ?thesis using iipEq topDen by simp
              qed
              from a22 tEq have vStop:"∃ xx. v = KStoptr xx" by simp
              then obtain xx where vDef:"v = KStoptr xx" by blast
              have a15Stop:"
                decl i_p (type.Storage x4) (Some (KStoptr (extractValueType v), type.Storage x4)) cp cd
(Memory st)
                  (Storage st (Address e_v)) (cd', mem', sck', e_v') = Some (c, m', k', e)"
                using a15 tEq Storage vDef by simp
              have topSome:"∃ pp. accessStore (ShowL_nat (Toploc sck')) k' = Some (KStoptr pp)"
                using cc1 lEq by blast
              have topEq:"accessStore (ShowL_nat (Toploc sck')) k' = Some (KStoptr (extractValueType
v))"
                using decl_stack_topLoc[OF a15Stop none topSome] .
              have "accessStore (ShowL_nat (Toploc sck')) k' = Some (KStoptr p)"
                using cc1 lEq by simp
              with topEq show ?thesis by simp
            qed
            have "t'' = x4"
```

```
            proof -
              have lEq:"l = ShowL_nat (Toploc sck')"
              proof -
                obtain tt where topDen:"Denvalue e $$ i_p = Some (tt, Stackloc (ShowL_nat (Toploc
sck')))"
                  using decl_stack_top[OF a15 none] by blast
                show ?thesis using iipEq topDen by simp
              qed
              have denTop:"Denvalue e $$ i_p = Some (type.Storage t'', Stackloc (ShowL_nat (Toploc
sck')))"
                using iipEq lEq by simp
              then obtain p0 where a3Eq:"Some (v, t) = Some (KStoptr p0, type.Storage t'')"
                using decl_storage_tp[OF a15 none] by blast
              then have "t = type.Storage t''" by simp
              with tEq show ?thesis by simp
            qed
            then show ?thesis using pVal stdef_fi inE CompStoType_sameLocNdTyp fmranI by fastforce
          next
            case False
            then have evp':"(Denvalue e_v') $$ ii = Some (type.Storage t'', Stackloc l)"
              using a15 decl_env_monotonic[OF a15] decl_env_not_i[OF a15] iidef by metis
            moreover have acc:"accessStore l sck' = accessStore l k'"
            proof -
              have ts':"TypeSafe e_v' (Accounts st) sck' mem' (Storage st) cd'" using as6 by blast
              have lSome:"∃sv. accessStore l sck' = Some sv"
                using ts' typeSafeAllStacklocsExist fmranI evp' by metis
              then have lNeTop:"l ≠ ShowL_nat (Toploc sck')"
                using ts' unfolding TypeSafe_def lessThanTopLocs_def LSubPrefL2_def
                by fastforce
              show ?thesis using decl_stack_change[OF a15 lNeTop] by simp
            qed
            ultimately obtain t2 l2 where t2Def:"((type.Storage t2, Storeloc l2) |∈| fmran (Denvalue
e_v') ∧ CompStoType t2 t'' l2 p)" using cc1 c''def fmranI by metis
            then have "(type.Storage t2, Storeloc l2) |∈| fmran (Denvalue e)" using
decl_env_monotonic[OF a15] fmranI by fast
            then show ?thesis using t2Def by metis
          qed
        qed
        ultimately show "fullyInitialised e (Accounts st) k'" unfolding fullyInitialised_def by blast
      qed
      have sameADD:"(Address e_v) = (Address e_v')" using as2 sameAddDecl ncp by blast

      from tEq obtain stloc tp'' where stdef:"
           (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
           (tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4 stloc
(extractValueType v))"
        using a28 by blast
      then have evd:"(type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v')"
      proof -
        obtain c_ev ct_ev dud_ev where ev_fi:"
             Type (Accounts st (Address e_v)) = Some (atype.Contract c_ev) ∧
             environment.Contract e_v = c_ev ∧
             ep $$ c_ev = Some (ct_ev, dud_ev) ∧
            (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
             (∀id vv. (ct_ev $$ id = Some (Var vv)) = (Denvalue e_v $$ id = Some (type.Storage vv,
Storeloc id)))"
          using as7 unfolding fullyInitialised_def by blast
        obtain c_ev' ct_ev' dud_ev' where ev'_fi:"
             Type (Accounts st (Address e_v')) = Some (atype.Contract c_ev') ∧
             environment.Contract e_v' = c_ev' ∧
             ep $$ c_ev' = Some (ct_ev', dud_ev') ∧
             (∀id v loc. Denvalue e_v' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
             (∀id vv. (ct_ev' $$ id = Some (Var vv)) = (Denvalue e_v' $$ id = Some (type.Storage vv,
Storeloc id)))"
```

```
        using as8 unfolding fullyInitialised_def by blast
      have eq_c:"c_ev = c_ev'" using ev_fi ev'_fi sameADD by simp
      hence eq_ct:"ct_ev = ct_ev'" using ev_fi ev'_fi eq_c by (metis option.inject prod.inject)
      from stdef obtain k_ev where k_def:"Denvalue e_v $$ k_ev = Some (type.Storage tp'', Storeloc
stloc)"
        using fmlookup_ran_iff by blast
      hence "ct_ev $$ stloc = Some (Var tp'')"
      proof -
        have ct_k:"ct_ev $$ k_ev = Some (Var tp'')" using ev_fi k_def by metis
        have den_k:"Denvalue e_v $$ k_ev = Some (type.Storage tp'', Storeloc k_ev)"
          using ev_fi ct_k by auto
        have "stloc = k_ev" using k_def den_k by simp
        thus ?thesis using ct_k by simp
      qed
      hence "Denvalue e_v' $$ stloc = Some (type.Storage tp'', Storeloc stloc)"
        using ev'_fi eq_ct by meson
      thus ?thesis using fmranI by meson
    qed

    have scon:"(∀ locs tp. SCon tp locs (state.Storage st (Address e_v)) ⟶ SCon tp locs
(state.Storage st (Address e_v')))"
      using sameADD by simp

    then have "
     (∃ loc tp'' p.
        (type.Storage tp'', loc) |∈| fmran (Denvalue e_v') ∧
        (case loc of
         Stackloc stloc ⇒
           accessStore stloc sck' = Some (KStoptr p) ∧
           (tp'' = x4 ∧ v = KStoptr p ∨ extractValueType v ≠ p ∧ CompStoType tp'' x4 p
(extractValueType v))
         | Storeloc stloc ⇒
             tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4
stloc (extractValueType v)))"
      using evd stdef by force

    then have a30:"TypeSafe e (Accounts st) k' m' (Storage st) c ∧
   (¬ cp ⟶
    (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
    Toploc (Memory st) ≤ Toploc m' ∧
    ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
    ncpOMemInDMem (Memory st) m' ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st)
m') ∧
   Toploc mem' ≤ Toploc m'"
      using typeSafeDecl[OF a2 a25 _ a15 ] as1 Storage a18 a26 as6 a27 a28 scon
      by (simp add:
          ‹local.load cp ((i_p, t_p) # pl) (ex # el) e_v' cd' sck' mem' e_v cd st g = Normal ((ev, cda,
k,
              m), g') ∧ (¬ cp ⟶ (∀ locs tp. MCon tp (state.Memory st) locs ⟶ MCon tp mem'
              locs) ∧ Toploc (state.Memory st) ≤ Toploc mem' ∧ ncpDenvalueLimit e_v' e_v sck' (Stack
st)
              (state.Memory st) ∧ ncpOMemInDMem (state.Memory st) mem' ∧ ncpElementsNoSubPref
(state.Memory st) mem' ∧
              ncpNewSelfPoint (state.Memory st) mem' ∧ Address e_v = Address e_v') ∧ TypeSafe e_v
(Accounts
              st) (Stack st) (state.Memory st) (state.Storage st) cd ∧ TypeSafe e_v' (Accounts st)
sck' mem' (state.Storage st)
              cd' ∧ fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e_v' (Accounts st)
sck'›
          ‹t = type.Storage x4›)

    have notSto:"¬ (case t_p of type.Storage x ⇒ cp | _ ⇒ False)"
      using Storage ncp by simp
    have cc1:"∀ ev cda k m g'.
```

```
       local.load cp pl el e c k' m' e_v cd st g4' = Normal ((ev, cda, k, m), g') ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m' locs) ∧
        Toploc (Memory st) ≤ Toploc m' ∧ ncpDenvalueLimit e e_v k' (Stack st) (Memory st) ∧
ncpOMemInDMem (Memory st) m'
            ∧ ncpElementsNoSubPref (Memory st) m' ∧ ncpNewSelfPoint (Memory st) m') ∧
       TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ TypeSafe e (Accounts st) k'
m' (Storage st) c ∧ Address e = Address e_v'
∧ fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e (Accounts st) k' ⟶
       TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       fullyInitialised ev (Accounts st) k
∧ (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧
        ncpElementsNoSubPref (Memory st) m ∧
        ncpNewSelfPoint (Memory st) m)"
        using "28.hyps"(2)[OF notSto none a10 _, of v t ] a15 Storage a2 as1 as2 as6 as7 as8 adds by
force
      then have cc2:"TypeSafe ev (Accounts st) k m (Storage st) cda ∧
       (¬ cp ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
        Toploc (Memory st) ≤ Toploc m ∧
        ncpDenvalueLimit ev e_v k (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) m ∧ ncpElementsNoSubPref (Memory st) m ∧ ncpNewSelfPoint (Memory st)
m)
∧ fullyInitialised ev (Accounts st) k "
        using a20 a2 as1 a30 as7 as8 fiE adds by blast
      moreover have "TypeSafe ev (Accounts st) k m (Storage st ) cda" using a20 a2 as1 a30 cc1 cc2 by
blast

      ultimately show "TypeSafe ev (Accounts st) k m (Storage st ) cda
∧ (¬ cp ⟶
          (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp m locs) ∧
          Toploc (Memory st) ≤ Toploc m ∧
          ncpElementsNoSubPref (Memory st) m ∧ ncpOMemInDMem (Memory st) m ∧
          ncpNewSelfPoint (Memory st) m ∧
          ncpDenvalueLimit ev e_v k (Stack st) (Memory st))
∧ fullyInitialised ev (Accounts st) k"
        using fmsubset_alt_def cc2 fmsubset_pred by meson
    qed
    then show "TypeSafe ev (Accounts st) k m (Storage st) cda" by simp
    then show "¬ cp ⟹ MCon tp (Memory st) locs ⟹ MCon tp m locs" using cc0 by simp
    show "¬ cp ⟹ Toploc (Memory st) ≤ Toploc m" using cc0 by simp
    show "¬ cp ⟹ ncpDenvalueLimit ev e_v k (Stack st) (Memory st)" using cc0 by simp
    show "¬ cp ⟹ ncpOMemInDMem (Memory st) m" using cc0 by simp
    show "¬ cp ⟹ ncpElementsNoSubPref (Memory st) m" using cc0 by simp
    show "¬ cp ⟹ ncpNewSelfPoint (Memory st) m " using cc0 by simp
    show "fullyInitialised ev (Accounts st) k" using cc0 by simp
  qed
next
  case (29 vt vu vv vw vx vy vz wa wb wc g)
  show ?case
  proof(intros)
    fix ev cd k m g' locs ls tp ct dud id' v
    assume a1: "local.load vt [] (vu # vv) vw vx vy vz wa wb wc g =
       Normal ((ev, cd, k, m), g') ∧
       (¬ vt ⟶
        (∀ locs tp. MCon tp (Memory wc) locs ⟶ MCon tp vz locs) ∧
        Toploc (Memory wc) ≤ Toploc vz ∧
        ncpDenvalueLimit vw wa vy (Stack wc) (Memory wc) ∧
        ncpOMemInDMem (Memory wc) vz ∧
        ncpElementsNoSubPref (Memory wc) vz ∧
```

```
        ncpNewSelfPoint (Memory wc) vz ∧ Address wa = Address vw) ∧
       TypeSafe wa (Accounts wc) (Stack wc) (Memory wc) (Storage wc) wb ∧
       TypeSafe vw (Accounts wc) vy vz (Storage wc) vx ∧
       fullyInitialised wa (Accounts wc) (Stack wc) ∧ fullyInitialised vw (Accounts wc) vy"
    then show "TypeSafe ev (Accounts wc) k m (Storage wc) cd" using load.simps(2) by (auto
split:if_split_asm result.splits)
    show "¬ vt ⟹ MCon tp (Memory wc) locs ⟹ MCon tp m locs" using a1 using load.simps(2) by simp
    show "¬ vt ⟹ Toploc (Memory wc) ≤ Toploc m" using a1 using load.simps(2) by simp
    show "¬ vt ⟹ ncpDenvalueLimit ev wa k (Stack wc) (Memory wc)" using a1 using load.simps by simp
    show "¬ vt ⟹ ncpOMemInDMem (Memory wc) m" using a1 using load.simps by simp
    show "¬ vt ⟹ ncpElementsNoSubPref (Memory wc) m" using a1 using load.simps by simp
    show "¬ vt ⟹ ncpNewSelfPoint (Memory wc) m " using a1 using load.simps by simp
    show "fullyInitialised ev (Accounts wc) k" using a1 using load.simps(2) by simp
  qed
next
  case (30 wd we wf wg wh wi wj wk wl wm g)
  show ?case
  proof(intros)
    fix ev cd k m g' locs ls tp ct dud id' v
    assume a1: "local.load wd (we # wf) [] wg wh wi wj wk wl wm g =
       Normal ((ev, cd, k, m), g') ∧
       (¬ wd ⟶
        (∀ locs tp. MCon tp (Memory wm) locs ⟶ MCon tp wj locs) ∧
        Toploc (Memory wm) ≤ Toploc wj ∧
        ncpDenvalueLimit wg wk wi (Stack wm) (Memory wm) ∧
        ncpOMemInDMem (Memory wm) wj ∧
        ncpElementsNoSubPref (Memory wm) wj ∧
        ncpNewSelfPoint (Memory wm) wj∧ Address wk = Address wg) ∧
       TypeSafe wk (Accounts wm) (Stack wm) (Memory wm) (Storage wm) wl ∧
       TypeSafe wg (Accounts wm) wi wj (Storage wm) wh ∧ fullyInitialised wk (Accounts wm) (Stack wm) ∧
fullyInitialised wg (Accounts wm) wi"
    then show "TypeSafe ev (Accounts wm) k m (Storage wm) cd" using load.simps(3) by (auto
split:if_split_asm result.splits)
    show "¬ wd ⟹ MCon tp (Memory wm) locs ⟹ MCon tp m locs" using a1 using load.simps(3) by simp
    show "¬ wd ⟹ Toploc (Memory wm) ≤ Toploc m" using a1 using load.simps(3) by simp
    show "¬ wd ⟹ ncpDenvalueLimit ev wk k (Stack wm) (Memory wm)" using a1 using load.simps by simp
    show "¬ wd ⟹ ncpOMemInDMem (Memory wm) m" using a1 using load.simps by simp
    show "¬ wd ⟹ ncpElementsNoSubPref (Memory wm) m" using a1 using load.simps by simp
    show "¬ wd ⟹ ncpNewSelfPoint (Memory wm) m " using a1 using load.simps by simp
    show "fullyInitialised ev (Accounts wm) k" using a1 using load.simps(3) by simp
  qed
next
  case (31 wn e_v' cd' sck' mem' e_v cd st g)
  show ?case
  proof(intros)
    fix ev cda k m g' locs ls tp ct dud id' v
    assume a1: "local.load wn [] [] e_v' cd' sck' mem' e_v cd st g =
       Normal ((ev, cda, k, m), g') ∧
       (¬ wn ⟶
        (∀ locs tp. MCon tp (Memory st) locs ⟶ MCon tp mem' locs) ∧
        Toploc (Memory st) ≤ Toploc mem' ∧
        ncpDenvalueLimit e_v' e_v sck' (Stack st) (Memory st) ∧
        ncpOMemInDMem (Memory st) mem' ∧
        ncpElementsNoSubPref (Memory st) mem' ∧
        ncpNewSelfPoint (Memory st) mem'∧ Address e_v = Address e_v') ∧
       TypeSafe e_v (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧
       TypeSafe e_v' (Accounts st) sck' mem' (Storage st) cd' ∧
       fullyInitialised e_v (Accounts st) (Stack st) ∧ fullyInitialised e_v' (Accounts st) sck'"
    then have fi1:"fullyInitialised e_v (Accounts st) (Stack st)" and fi2:"fullyInitialised e_v'
(Accounts st) sck'" by blast+
    then show "TypeSafe ev (Accounts st) k m (Storage st) cda" using load.simps(4)[of wn e_v' cd' sck'
mem' e_v cd st g] a1 by (auto split:if_split_asm result.splits)
    show "¬ wn ⟹ MCon tp (Memory st) locs ⟹ MCon tp m locs" using a1 using load.simps(4) by auto
    show "¬ wn ⟹ Toploc (Memory st) ≤ Toploc m" using a1 using load.simps(4) by auto
```

```
    show "¬ wn ⟹ ncpDenvalueLimit ev e_v k (Stack st) (Memory st)" using a1 using load.simps(4) by
auto
    show "¬ wn ⟹ ncpOMemInDMem (Memory st) m" using a1 using load.simps(4) by auto
    show "¬ wn ⟹ ncpElementsNoSubPref (Memory st) m" using a1 using load.simps(4) by auto
    show "¬ wn ⟹ ncpNewSelfPoint (Memory st) m " using a1 using load.simps(4) by auto
    have eq:"(e_v', cd', sck', mem') = (ev, cda, k, m)" using a1 load.simps(4) by simp
    then show "fullyInitialised ev (Accounts st) k" using fi2 by simp
  qed
next
  case (32 i e cd st g)
  show ?case
  proof(intros)
    fix v3' t3' g3' assume a1:"rexp (l.Id i) e cd st g = Normal (( v3', t3'), g3') ∧ TypeSafe e
(Accounts st) (Stack st) (Memory st) (Storage st) cd
                                ∧ fullyInitialised e (Accounts st) (Stack st)"
    then consider
      (Stack) tp l where "fmlookup (Denvalue e) i = Some (tp, Stackloc l)"
    |(store) tp l where "fmlookup (Denvalue e) i = Some (tp, Storeloc l)" using rexp.simps(1) a1
      by (simp split:option.splits prod.splits type.splits denvalue.splits stypes.splits)
    then show "exprResultCon e cd st v3' t3'"
    proof (cases)
      case Stack
      then have a7:"(tp, Stackloc l) |∈| fmran (Denvalue e)" using Finite_Map.fmranI by fast
      then consider
        (kval) v where "accessStore l (Stack st) = Some(KValue v)"
      | (cdptr) p where "accessStore l (Stack st) = Some (KCDptr p)"
      | (memptr) p where "accessStore l (Stack st) = Some (KMemptr p)"
      | (stoptr) p where "accessStore l (Stack st) = Some (KStoptr p)"
        using rexp.simps(1) a1 Stack by (simp split:option.splits prod.splits type.splits
denvalue.splits stypes.splits stackvalue.splits )
      then show "exprResultCon e cd st v3' t3'"
      proof(cases)
        case kval
        then show ?thesis using a1 Stack a7 unfolding rexp.simps(1) TypeSafe_def typeCompat_def by
(cases tp; force)
      next
        case cdptr
        then show ?thesis using a1 Stack a7 unfolding TypeSafe_def rexp.simps(1) typeCompat_def by
(cases tp; force)
      next
        case memptr
        show ?thesis
        proof(cases tp)
          case (Value x1)
          then show ?thesis using a1 Stack a7 memptr unfolding TypeSafe_def typeCompat_def
rexp.simps(1) by force
        next
          case (Calldata x2)
          then show ?thesis using a1 Stack a7 memptr unfolding TypeSafe_def rexp.simps(1)
typeCompat_def by force
        next
          case (Memory x3)
          then have a10:"(v3', t3') = (KMemptr p, type.Memory x3)" using a1 Stack rexp.simps(1)[of i e
cd st g ] Stack a7 memptr
            by (auto split:option.splits type.splits mtypes.splits stypes.splits)
          then have a20:"MCon x3 (Memory st) (extractValueType v3')" using a1 Stack a7 memptr Memory
unfolding rexp.simps(1) TypeSafe_def typeCompat_def by force
          have a30:"(∃xx. v3' = KMemptr xx)" using a1 Stack rexp.simps(1) a7 memptr Memory by auto
          have a35:"extractValueType v3' = p" using a10 by simp
          then show ?thesis using a20 a30 a1 Stack a7 memptr Memory a35 unfolding rexp.simps(1) by
force
        next
          case (Storage x4)
          then show ?thesis using a1 Stack a7 memptr unfolding TypeSafe_def rexp.simps(1)
```

```
typeCompat_def by force
        qed
      next
        case stoptr
        then show ?thesis
        proof (cases tp)
          case (Value x1)
          then show ?thesis using a1 stoptr Stack a7 unfolding TypeSafe_def rexp.simps(1)
typeCompat_def by force
        next
          case (Calldata x2)
          then show ?thesis using a1 stoptr Stack a7 unfolding TypeSafe_def typeCompat_def by force
        next
          case (Memory x3)
          then show ?thesis using a1 stoptr Stack a7 unfolding TypeSafe_def typeCompat_def by force
        next
          case (Storage x4)
          then have a30: "t3' = type.Storage x4" and a40:"v3' = KStoptr p" using a1 Stack stoptr un-
folding rexp.simps(1)
            by (simp split:option.splits prod.splits type.splits denvalue.splits stypes.splits
stackvalue.splits)+
          have "SCon x4 (extractValueType v3') (Storage st (Address e)) ∧
        (∃xx. v3' = KStoptr xx) ∧
        (∃stloc tp''.
            (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
            (tp'' = x4 ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧ CompStoType tp'' x4
stloc (extractValueType v3')))"
          proof intros
            show "SCon x4 (extractValueType v3') (Storage st (Address e))" using stoptr Stack a7
Storage a30 a40 using a1 unfolding TypeSafe_def typeCompat_def by force
            show "∃xx. v3' = KStoptr xx" using a40 by simp
            show "∃stloc tp''.
       (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
       (tp'' = x4 ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧ CompStoType tp'' x4 stloc
(extractValueType v3'))"
              using a1 unfolding fullyInitialised_def
              by (metis CompStoType_sameLoc_sameType Storage a40 a7 extractValueType.simps(4) stoptr)
          qed
          then show ?thesis using a1 stoptr Stack a7 Storage a30 a40 unfolding TypeSafe_def
typeCompat_def by simp
        qed
      qed
    next
      case store
      then have a7:"(tp, Storeloc l) |∈| fmran (Denvalue e)" using Finite_Map.fmranI store by simp
      then consider
        (stval) t where "tp = type.Storage (STValue t)"
      | (stary) t x where "tp = type.Storage (STArray x t)"
        using rexp.simps(1) a1 store by (simp split:option.splits prod.splits type.splits
denvalue.splits stypes.splits stackvalue.splits)

      then show "exprResultCon e cd st v3' t3'"
      proof(cases)
        case stval
        then have a20:"accessStorage t l (Storage st (Address e)) = (case (Storage st (Address e)) $$ l
of
              Some v ⇒ v
            | None ⇒ ival t)" using accessStorage_def by auto
        then show ?thesis
        proof(cases "(Storage st (Address e)) $$ l")
          case None
          then have "accessStorage t l (Storage st (Address e)) = ival t" using a20 by simp
          then have "KValue (ival t) = v3'" using a20 stval store rexp.simps(1) None a1 by simp
          then show ?thesis using ivalTypeCon[of t "extractValueType v3'"] a20 stval store
```

```
rexp.simps(1) None a1 by auto
        next
          case (Some a)
          then have a30:"SCon (STValue t) l ((Storage st) (Address e))" using a20 stval store a1 a7
unfolding TypeSafe_def typeCompat_def by force
          moreover have "SCon (STValue t) l ((Storage st) (Address e)) = (typeCon t a)" using Some
SCon.simps(1) accessStorage_def by simp
          moreover have "extractValueType v3' = a" using a20 stval store a1 a7 rexp.simps(1) Some by
auto
          ultimately show ?thesis using stval store a1 rexp.simps(1) by auto
        qed
      next
        case stary
        then have a30: "t3' = type.Storage (STArray x t)" and "v3' = KStoptr l" using a1 store un-
folding rexp.simps(1)
          by (simp split:option.splits prod.splits type.splits denvalue.splits stypes.splits
stackvalue.splits)+
        then have a35:"SCon (STArray x t) l ((Storage st) (Address e))" using store a1 a7 stary un-
folding TypeSafe_def typeCompat_def by force
        then show ?thesis using stary store a1 a30 unfolding rexp.simps(1)
          using a7 by auto
      qed
    qed
  qed
next
  case (33 i r e cd st g)
  show ?case
  proof(intros)
    fix v3' t3' g3' assume a1:" local.rexp (Ref i r) e cd st g = Normal ((v3', t3'), g3') ∧
       TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd
       ∧ fullyInitialised e (Accounts st) (Stack st) "
    then consider
      (stloc) tp l where "fmlookup (Denvalue e) i = Some (tp, Stackloc l)"
    |(stoloc) tp l where "fmlookup (Denvalue e) i = Some (tp, Storeloc l)"
      using a1 by (simp add: rexp.simps split:option.split_asm prod.split_asm denvalue.split_asm)
    then show "exprResultCon e cd st v3' t3'"
    proof (cases)
      case stloc
      then have a7:"(tp, Stackloc l) |∈| fmran (Denvalue e)" using Finite_Map.fmranI[of "Denvalue e"]
by simp
      then consider
        (cdptr) l' where "accessStore l (Stack st) = Some (KCDptr l')"
      | (memptr) l' where "accessStore l (Stack st) = Some (KMemptr l')"
      | (stoptr) l' where "accessStore l (Stack st) = Some (KStoptr l')"
        using a1 stloc by (simp add: rexp.simps split:option.split_asm prod.split_asm
denvalue.split_asm stackvalue.split_asm)
      then show ?thesis
      proof(cases)
        case cdptr
        then obtain t where a10:"tp = Calldata t" using stloc a1 by (simp add: rexp.simps
split:option.split_asm prod.split_asm denvalue.split_asm stackvalue.split_asm type.split_asm)
        then obtain l'' t' g' where a20:"msel False t l' r e cd st g = Normal ((l'',t'), g')"
          using stloc a1 cdptr by (simp add: rexp.simps split:option.split_asm prod.split_asm
denvalue.split_asm stackvalue.split_asm type.split_asm mtypes.split_asm result.split_asm)
        then have a22:"r ≠ []" using msel.simps by auto

        have a25:"MCon t cd l'" using a1 a7 cdptr stloc a10 a1 unfolding TypeSafe_def typeCompat_def
          by (auto split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm stackvalue.split_asm type.split_asm )

        then show ?thesis
        proof(cases t')
          case (MTArray x11 t'')
          then obtain p where a30:"accessStore l'' cd = Some(MPointer p)" using MTArray stloc a1 cdptr
```

```
a10 a20 rexp.simps(2)[of i r e cd st
              g ] by (auto split:option.split_asm prod.split_asm denvalue.split_asm
stackvalue.split_asm type.split_asm mtypes.split_asm
              result.split_asm memoryvalue.split_asm)

          then have a40:"v3' = KCDptr p" using stloc a1 cdptr a10 a20 by (simp add:rexp.simps(2)
split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm)
          moreover have a45:"t3' = Calldata (MTArray x11 t'')" using stloc a1 cdptr a10 a20 MTArray
by (simp add:rexp.simps(2)
              split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm)
          moreover have a60:"r ≠ [] ⟶
       (∃len arr.
           t = MTArray len arr ∧
           (case t' of
            MTArray l'a ar' ⇒
              ∃p. accessStore l'' (if False then Memory st else cd) = Some (MPointer p) ∧
                  CompMemType (if False then Memory st else cd) len arr t' l' p
            | MTValue val ⇒
                CompMemType (if False then Memory st else cd) len arr t' l' l''))"
            using "33.hyps"(1)[of "(tp, Stackloc l)" tp "Stackloc l" l "KCDptr l'" l' t g]
              stloc cdptr a10 a20 a30 a25 a1 by simp
          then obtain len arr where a35:"t = MTArray len arr" using a25 MTArray
              using a22 by blast
          moreover have "CompMemType cd len arr t' l' p"
              using MTArray a40 extractValueType.simps(2) calculation a22 a30 a25 a35 a60 by simp
          moreover have "MCon (MTArray x11 t'') cd (extractValueType v3')"
              using MTArray a40 extractValueType.simps(2) calculation
              using a22 a30 a25 a35 CompTypeRemainsMCon by presburger
          moreover have a70:"(∃len arr.
              t = MTArray len arr ∧ (∃p. accessStore l'' (if False then Memory st else cd) = Some
(MPointer p) ∧ CompMemType (if False then Memory st else cd) len arr t' l' p))"
              using a60 a20 a22 MTArray by auto
          moreover have a80:"(Calldata t, Stackloc l) |∈| fmran (Denvalue e)" using a10 a7 by blast
          moreover have a90:"accessStore l (Stack st) = Some (KCDptr l')" using cdptr by blast
          moreover have a100:"extractValueType v3' = p" using a40 by auto
          ultimately have " MCon (MTArray x11 t'') cd (extractValueType v3') ∧
        (∃xx. v3' = KCDptr xx) ∧
        (∃stloc tp'' p.
            (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e) ∧
            accessStore stloc (Stack st) = Some (KCDptr p) ∧
            (tp'' = (MTArray x11 t'') ∧ v3' = KCDptr p ∨ (∃len arr. extractValueType v3' ≠ p ∧
                tp'' = MTArray len arr ∧ CompMemType cd len arr (MTArray x11 t'') p (extractValueType
v3'))))"
              by (metis (no_types, lifting) BothMConImpsNotCompMemType MTArray memoryvalue.inject(2) a25
a30 option.inject)
          then show ?thesis using a45
            by (auto split:type.splits if_splits)
        next
          case (MTValue x2)
          then obtain v where a30:"accessStore l'' cd = Some(MValue v)" using MTValue stloc a1 cdptr
a10 a20 rexp.simps(2)[of i r e cd st g
              ] by (auto split:option.split_asm prod.split_asm denvalue.split_asm stackvalue.split_asm
type.split_asm mtypes.split_asm
              result.split_asm memoryvalue.split_asm)
          then have a40:"v3' = KValue v" using stloc a1 cdptr a10 a20 by (simp add:rexp.simps(2)
split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm)
          moreover have "t3' = Value x2" using stloc a1 cdptr a10 a20 MTValue by (simp
add:rexp.simps(2) split:option.split_asm
              prod.split_asm denvalue.split_asm mtypes.split_asm memoryvalue.split_asm)
          moreover have a60:"MCon t' cd l''"
            using "33.hyps"(1)[of "(tp, Stackloc l)" tp "Stackloc l" l "KCDptr l'" l' t g]
              stloc cdptr a10 a20 a30 a25 a1 a22 a30 MTValue
            by (metis (no_types, lifting) CompTypeRemainsMCon mtypes.simps(6) type.simps(18)
```

```
return.simps)
          then have "typeCon x2 (extractValueType v3')" using extractValueType.simps(1)[of v] a40 a60
a30 MTValue by simp
          ultimately show ?thesis by simp
        qed
      next
        case memptr
        then obtain t where a10:"tp = type.Memory t" using stloc a1
          by (simp add: rexp.simps split:option.split_asm prod.split_asm denvalue.split_asm
stackvalue.split_asm type.split_asm)
        then obtain l'' t' g' where a20:"msel True t l' r e cd st g = Normal ((l'',t'), g')"
          using stloc a1 memptr by (simp add: rexp.simps split:option.split_asm prod.split_asm
denvalue.split_asm stackvalue.split_asm type.split_asm mtypes.split_asm result.split_asm)
        then have a22:"r ≠ []" using msel.simps by auto
        have a25:"MCon t (Memory st) l'" using a1 a7 memptr stloc a10 a1 unfolding TypeSafe_def
typeCompat_def by (auto
            split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm stackvalue.split_asm
            type.split_asm )
        then show ?thesis
        proof(cases t')
          case (MTArray x11 t'')
          then obtain p where a30:"accessStore l'' (Memory st) = Some(MPointer p)" using MTArray stloc
a1 memptr a10 a20 rexp.simps(2)[of i
              r e cd st g ] by (auto split:option.split_asm prod.split_asm denvalue.split_asm
stackvalue.split_asm type.split_asm
              mtypes.split_asm result.split_asm memoryvalue.split_asm)
          then have a40:"v3' = KMemptr p" using stloc a1 memptr a10 a20 by (simp add:rexp.simps(2)
split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm)
          moreover have a45:"t3' = type.Memory (MTArray x11 t'')" using stloc a1 memptr a10 a20
MTArray by (simp add:rexp.simps(2)
              split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm)
          moreover have a60:"r ≠ [] ⟶
       (∃len arr.
           t = MTArray len arr ∧
           (case t' of
            MTArray l'a ar' ⇒
              ∃p. accessStore l'' (if True then Memory st else cd) = Some (MPointer p) ∧
                  CompMemType (if True then Memory st else cd) len arr t' l' p
            | MTValue val ⇒ CompMemType (if True then Memory st else cd) len arr t' l' l''))"
            using "33.hyps"(2)[of "(tp, Stackloc l)" tp "Stackloc l" l "KMemptr l'" l' t g] stloc
memptr a10 a20 a30 a25 a1 by simp+
          then obtain len arr where a35:"t = MTArray len arr" using a25 MTArray
            using a22 by blast
          then have a65:"CompMemType (Memory st) len arr t' l' p "
            using MTArray a40 extractValueType.simps(2) calculation a22 a30 a60 a35 a25 by simp
          moreover have "MCon (MTArray x11 t'') (Memory st) (extractValueType v3')"
            using MTArray a40 extractValueType.simps(2) calculation a22 a30 a60 a35 a25 a65
            using CompTypeRemainsMCon extractValueType.simps(3) by presburger
          moreover have a70:"(∃len arr.
            t = MTArray len arr ∧ (∃p. accessStore l'' (if True then Memory st else cd) = Some
(MPointer p) ∧ CompMemType (if True then Memory st else cd) len arr t' l' p))"
            using a60 a20 a22 MTArray by auto
          moreover have a80:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e)" using a10 a7 by auto
          moreover have a90:"accessStore l (Stack st) = Some (KMemptr l')" using memptr by blast
          moreover have a100:"extractValueType v3' = p" using a40 by auto

          ultimately have " MCon (MTArray x11 t'') (Memory st) (extractValueType v3') ∧
        (∃xx. v3' = KMemptr xx) ∧
        (∃stloc tp'' p.
            (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e) ∧
            accessStore stloc (Stack st) = Some (KMemptr p) ∧
            (tp'' = (MTArray x11 t'') ∧ v3' = KMemptr p ∨ (∃len arr. extractValueType v3' ≠ p ∧
```

```
              tp'' = MTArray len arr ∧ CompMemType (Memory st) len arr (MTArray x11 t'') p
(extractValueType v3'))))"
          by (metis (no_types, lifting) BothMConImpsNotCompMemType MTArray memoryvalue.inject(2) a25
a30 option.inject)
        then show ?thesis using a45 by (auto split:type.splits if_splits)
      next
        case (MTValue x2)
        then obtain v where a30:"accessStore l'' (Memory st) = Some(MValue v)"
          using MTValue stloc a1 memptr a10 a20 rexp.simps(2)[of i r e cd st g ]
          by (auto split:option.split_asm prod.split_asm denvalue.split_asm stackvalue.split_asm
type.split_asm mtypes.split_asm result.split_asm memoryvalue.split_asm)
        then have a40:"v3' = KValue v"
          using stloc a1 memptr a10 a20
          by (simp add:rexp.simps(2) split:option.split_asm prod.split_asm denvalue.split_asm
mtypes.split_asm)
        moreover have "t3' = Value x2"
          using stloc a1 memptr a10 a20 MTValue
          by (simp add:rexp.simps(2) split:option.split_asm prod.split_asm denvalue.split_asm
mtypes.split_asm memoryvalue.split_asm)
        moreover have a60:"MCon t' (Memory st) l''"
          using "33.hyps"(2)[of "(tp, Stackloc l)" tp "Stackloc l" l "KMemptr l'" l' t g]
            stloc memptr a10 a20 a30 a25 a1 MTValue a22
          by (metis (lifting) mtypes.simps(6) type.case(3) return.simps CompTypeRemainsMCon)
        then have "typeCon x2 (extractValueType v3')" using extractValueType.simps(1)[of v] a40 a60
a30 MTValue by simp
        ultimately show ?thesis by simp
      qed
    next
      case stoptr
      then obtain t where a10:"tp = type.Storage t" using stloc a1
        by (simp add: rexp.simps split:option.split_asm prod.split_asm denvalue.split_asm
stackvalue.split_asm type.split_asm)
      then obtain l'' t' g' where a20:"ssel t l' r e cd st g = Normal ((l'',t'), g')"
        using stloc a1 stoptr
        by (simp add: rexp.simps split:option.split_asm prod.split_asm denvalue.split_asm
stackvalue.split_asm
            type.split_asm mtypes.split_asm result.split_asm)
      have a25:"SCon t l' (Storage st (Address e))"
        using a1 a7 stoptr stloc a10 a1
        unfolding TypeSafe_def typeCompat_def
        by (auto split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm stackvalue.split_asm type.split_asm )
      then show ?thesis
      proof(cases t')
        case (STArray x11 t'')
        have ret:"(KStoptr l'', type.Storage t') = (v3', t3')" using a1 rexp.simps(2)[of i r e cd st
g ] stoptr stloc a10 a20 STArray
          by (simp split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm)

        have "(t = t' ∧ v3' = KStoptr l' ∨ extractValueType v3' ≠ l' ∧ CompStoType t t' l'
(extractValueType v3'))"
        proof(cases "r = []")
          case True
          then have "l'' = l'" using a20 ssel.simps by simp
          moreover have "t' = t" using a20 ssel.simps(1)[of t l' e cd st g] True by simp
          ultimately show ?thesis using ret by simp
        next
          case False
          then have "CompStoType t t' l' l''"
            using "33.hyps"(3)[of "(tp, Stackloc l)" tp "Stackloc l" l "KStoptr l'" l' t g] stloc
stoptr a10 a20 a25 a1 STArray by simp
          then show ?thesis using ret
            using CompStoType_sameLoc_sameType by force
```

```
          qed
          then have "SCon t' (l'') (Storage st (Address e))" using a25 SCon_imps_sublocs ret by
fastforce
          then have g1:"SCon (STArray x11 t'') (extractValueType v3') (Storage st (Address e))" using
STArray ret by auto
          have "SCon (STArray x11 t'') (extractValueType v3') (Storage st (Address e)) ∧
        (∃xx. v3' = KStoptr xx) ∧
        (∃stloc tp''.
            (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
            (tp'' = (STArray x11 t'') ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧
CompStoType tp'' (STArray x11 t'') stloc (extractValueType v3')))"

          proof intros
            show " SCon (STArray x11 t'') (extractValueType v3') (Storage st (Address e))" using g1 by
simp
            show "∃xx. v3' = KStoptr xx" using ret by blast
            show "∃stloc tp''.
       (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
       (tp'' = STArray x11 t'' ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧ CompStoType tp''
(STArray x11 t'') stloc (extractValueType v3')) "
            proof -
              obtain tp0 s0 where "(type.Storage tp0, Storeloc s0) |∈| fmran (Denvalue e)"
"CompStoType tp0 t s0 l'"
                using a1 a7 a10 stoptr unfolding fullyInitialised_def by blast
              then show ?thesis
                using CompStoType_ptr_witness ‹∃xx. v3' = KStoptr xx›
                  ‹t = t' ∧ v3' = KStoptr l' ∨ extractValueType v3' ≠ l' ∧ CompStoType t t' l'
(extractValueType v3')›
                  STArray by blast
            qed
          qed
          then show ?thesis using a20 a10 stoptr a7 g1 ret STArray stloc by fastforce
        next
          case (STMap x21 x22)
          then have a40:"v3' = KStoptr l''"
            using stloc a1 stoptr a10 a20 by (simp add:rexp.simps(2) split:option.split_asm
prod.split_asm denvalue.split_asm mtypes.split_asm)
          moreover have a50:"t3' = type.Storage (STMap x21 x22)"
            using stloc a1 stoptr a10 a20 STMap
            by (simp add:rexp.simps(2) split:option.split_asm prod.split_asm denvalue.split_asm
mtypes.split_asm memoryvalue.split_asm)
          have "(t = t' ∧ v3' = KStoptr l' ∨ extractValueType v3' ≠ l' ∧ CompStoType t t' l'
(extractValueType v3'))"
          proof(cases "r = []")
            case True
            then have "l'' = l'" using a20 ssel.simps by simp
            moreover have "t' = t" using a20 ssel.simps(1)[of t l' e cd st g] True by simp
            ultimately show ?thesis using a40 a50 by simp
          next
            case False
            then have "CompStoType t t' l' l''"
              using "33.hyps"(3)[of "(tp, Stackloc l)" tp "Stackloc l" l "KStoptr l'" l' t g] stloc
stoptr a10 a20 a25 a1 STMap by simp
            then show ?thesis using a40 a50 using CompStoType_sameLoc_sameType by force
          qed
          then have "SCon t' (l'') (Storage st (Address e))" using a25
            using SCon_imps_sublocs calculation by fastforce
          then have g1:"SCon (STMap x21 x22) (extractValueType v3') (Storage st (Address e))" using
STMap a40 by simp

          have "SCon (STMap x21 x22) (extractValueType v3') (Storage st (Address e)) ∧
        (∃xx. v3' = KStoptr xx) ∧
        (∃stloc tp''.
            (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
```

```
          (tp'' = (STMap x21 x22) ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧ CompStoType
tp'' (STMap x21 x22) stloc (extractValueType v3')))"

          proof intros
            show " SCon (STMap x21 x22) (extractValueType v3') (Storage st (Address e))" using g1 by
simp
            show "∃xx. v3' = KStoptr xx" using a40 by auto
            show "∃stloc tp''.
       (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e) ∧
       (tp'' = STMap x21 x22 ∧ v3' = KStoptr stloc ∨ extractValueType v3' ≠ stloc ∧ CompStoType tp''
(STMap x21 x22) stloc (extractValueType v3')) "
            proof -
              obtain tp0 s0 where "(type.Storage tp0, Storeloc s0) |∈| fmran (Denvalue e)"
"CompStoType tp0 t s0 l'"
                using a1 a7 a10 stoptr unfolding fullyInitialised_def by blast
              then show ?thesis
                using CompStoType_ptr_witness ‹∃xx. v3' = KStoptr xx›
                  ‹t = t' ∧ v3' = KStoptr l' ∨ extractValueType v3' ≠ l' ∧ CompStoType t t' l'
(extractValueType v3')›
                  STMap by blast
            qed
          qed

          then show ?thesis using a20 a10 stoptr a7 g1 a40 a50 STMap stloc by force
        next
          case (STValue t'')
          then have a40:"v3' = KValue (accessStorage t'' l'' (Storage st (Address e)))" using stloc a1
stoptr a10 a20 by (simp
              add:rexp.simps(2) split:option.split_asm prod.split_asm denvalue.split_asm
mtypes.split_asm)
          moreover have "t3' = Value t''" using stloc a1 stoptr a10 a20 STValue by (simp
add:rexp.simps(2) split:option.split_asm
              prod.split_asm denvalue.split_asm mtypes.split_asm memoryvalue.split_asm)
          moreover have a60:"SCon t (l') (Storage st (Address e))" using "33.hyps"(3) stloc stoptr a10
a20 a25 a1 STValue by simp
          then have cc0:"(t = t' ∧ l' = l'' ∨ CompStoType t t' l' l'')"
          proof(cases "r = []")
            case True
            then have "l'' = l'" using a20 ssel.simps by auto
            moreover have "t' = t" using a20 ssel.simps(1)[of t l' e cd st g] True by simp
            ultimately show ?thesis using a40 by simp
          next
            case False
            then have "CompStoType t t' l' l''"
              using a10 a20 a25 a1 STValue "33.hyps"(3) stoptr
              by (simp add: stloc)
            then show ?thesis using a40 using CompStoType_sameLoc_sameType by simp
          qed
          then have "typeCon t'' (accessStorage t'' l'' (Storage st (Address e)))"
            using a60 SCon_imps_sublocs STValue SCon.simps(1) by blast
          ultimately show ?thesis by simp
        qed
      qed
    next
      case stoloc
      then have a7:"(tp, Storeloc l) |∈| fmran (Denvalue e)" using Finite_Map.fmranI[of "Denvalue e"]
by simp

      then obtain t where a10:"tp = type.Storage t" using stoloc a1 by (simp add: rexp.simps
split:option.split_asm prod.split_asm denvalue.split_asm stackvalue.split_asm type.split_asm)
      then obtain l'' t' g' where a20:"ssel t l r e cd st g = Normal ((l'',t'), g')"
        using a1 stoloc by (simp add: rexp.simps split:option.split_asm prod.split_asm
denvalue.split_asm stackvalue.split_asm type.split_asm mtypes.split_asm result.split_asm)
      have a25:"SCon t l (Storage st (Address e))" using a1 a7 stoloc a10 a1 unfolding TypeSafe_def
```

```
typeCompat_def by (auto
          split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm stackvalue.split_asm
          type.split_asm )
      then show ?thesis
      proof(cases t')
        case (STArray x11 t'')
        then have a40:"v3' = KStoptr l''"
          using a1 stoloc a10 a20 by (simp add:rexp.simps(2) split:option.split_asm prod.split_asm
denvalue.split_asm mtypes.split_asm)
        moreover have a50:"t3' = type.Storage (STArray x11 t'')"
          using stoloc a1 a10 a20 STArray
          by (simp add:rexp.simps(2) split:option.split_asm prod.split_asm denvalue.split_asm
mtypes.split_asm memoryvalue.split_asm)
        have cc0:"(t = t' ∧ v3' = KStoptr l ∨ extractValueType v3' ≠ l ∧ CompStoType t t' l
(extractValueType v3'))"
        proof(cases "r = []")
          case True
          then have "l'' = l" using a20 ssel.simps by simp
          moreover have "t' = t" using a20 ssel.simps(1)[of t l e cd st g] True by simp
          ultimately show ?thesis using a40 a50 by simp
        next
          case False
          then have "CompStoType t t' l l''"
            using stoloc a10 a20 a25 a1 STArray "33.hyps"(4) by auto
          then show ?thesis using a40 a50 using CompStoType_sameLoc_sameType by force
        qed
        then have "SCon t' (l'') (Storage st (Address e))" using a25
          by (metis SCon_imps_sublocs calculation extractValueType.simps(4))
        then have g1:"SCon (STArray x11 t'') (extractValueType v3') (Storage st (Address e))" using
STArray a40 by simp
        then show ?thesis using a20 a10 stoloc a7 g1 a40 a50 STArray cc0 by force
      next
        case (STMap x21 x22)
        then have a40:"v3' = KStoptr l''" using stoloc a1 a10 a20 by (simp add:rexp.simps(2)
split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm)
        moreover have a50:"t3' = type.Storage (STMap x21 x22)" using stoloc a1 a10 a20 STMap by (simp
add:rexp.simps(2)
            split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm
memoryvalue.split_asm)
        have cc0:"(t = t' ∧ v3' = KStoptr l ∨ extractValueType v3' ≠ l ∧ CompStoType t t' l
(extractValueType v3'))"
        proof(cases "r = []")
          case True
          then have "l'' = l" using a20 ssel.simps by simp
          moreover have "t' = t" using a20 ssel.simps(1)[of t l e cd st g] True by simp
          ultimately show ?thesis using a40 a50 by simp
        next
          case False
          then have "CompStoType t t' l l''"
            using stoloc a10 a20 a25 a1 STMap "33.hyps"(4) by auto
          then show ?thesis using a40 a50 using CompStoType_sameLoc_sameType by force
        qed
        then have "SCon t' (l'') (Storage st (Address e))" using a25
          using SCon_imps_sublocs calculation by fastforce
        then have g1:"SCon (STMap x21 x22) (extractValueType v3') (Storage st (Address e))" using
STMap a40 by simp
        then show ?thesis using a20 a10 stoloc a7 g1 a40 a50 STMap cc0 by auto
      next
        case (STValue t'')
        then have a40:"v3' = KValue (accessStorage t'' l'' (Storage st (Address e)))" using stoloc a1
a10 a20 by (simp add:rexp.simps(2)
            split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm)
        moreover have "t3' = Value t''" using stoloc a1 a10 a20 STValue by (simp add:rexp.simps(2)
```

```
split:option.split_asm prod.split_asm denvalue.split_asm mtypes.split_asm memoryvalue.split_asm)
        moreover have a60:"SCon t (l) (Storage st (Address e))" using "33.hyps"(4) stoloc a10 a20 a25
a1 STValue by simp
        then have "r ≠ [] ⟶ CompStoType t t' l l''"
          using "33.hyps"(4)[OF stoloc, of tp "Storeloc l" l t g] stoloc a10 a20 a25 a1 STValue a40 by
simp
        then have cc0:"(t = t' ∧ l =l''
                        ∨ CompStoType t t' l l'')"
        proof(cases "r = []")
          case True
          then have "l'' = l" using a20 ssel.simps by simp
          moreover have "t' = t" using a20 ssel.simps(1)[of t l e cd st g] True by simp
          ultimately show ?thesis using a40 by simp
        next
          case False
          then have "CompStoType t t' l l''"
            using stoloc a10 a20 a25 a1 STValue "33.hyps"(4) by auto
          then show ?thesis using a40 using CompStoType_sameLoc_sameType by simp
        qed
        then have "typeCon t'' (accessStorage t'' l'' (Storage st (Address e)))" using a60
          using SCon_imps_sublocs STValue SCon.simps(1) by blast
        ultimately show ?thesis by simp
      qed
    qed
  qed
next
  case (34 e cd st g)
  show ?case
  proof (intros)
    fix v t g4' assume a1:"local.expr CONTRACTS e cd st g = Normal ((v, t), g4') ∧
       TypeSafe e (Accounts st) (Stack st) (Memory st) (Storage st) cd ∧ fullyInitialised e (Accounts
st) (Stack st)"
    then have "t = Value TAddr" using a1 expr.simps(20)[of e cd st g ] by (simp split:if_split_asm
result.split_asm prod.split_asm )
    moreover obtain n where a2:"(Contracts (Accounts st (Address e))) = Suc n" using a1
expr.simps(20) by (simp split:if_splits result.splits prod.splits nat.splits)
    moreover have vdef:"v = KValue (hash_version (Address e) (ShowL_nat n))" using expr.simps(20)[of e
cd st g ] a1 a2
      by (auto split:if_split_asm result.splits prod.split_asm nat.splits)
    moreover have "typeCon TAddr (extractValueType v)" unfolding typeCon.simps
    proof -
      have in1:"extractValueType v = (hash_version (Address e) (ShowL_nat n))"
        using vdef extractValueType.simps by auto
      then have in2:"extractValueType v = (ShowL_nat (n)) + (STR ''-'' + Address e)" unfolding
hash_version_def
        by blast
      have "CHR ''.'' ∉ set (literal.explode (Address e))"
        using a1 unfolding TypeSafe_def addressFormat_def typeCon.simps checkAddress_def by blast
      then have "CHR ''.'' ∉ set (literal.explode (STR ''-'' + Address e))"
        by (simp add: Literal.rep_eq add_Literal_assoc)
      moreover have "CHR ''.'' ∉ set (literal.explode (ShowL_nat (n)))" using ShowLNatDot by blast

      ultimately have "CHR ''.'' ∉ set (literal.explode ((ShowL_nat (n)) + (STR ''-'' + Address e)))"
        by (simp add: plus_literal.rep_eq)
      then show "checkAddress (extractValueType v) " using in2 unfolding checkAddress_def by auto

    qed


    ultimately show "exprResultCon e cd st v t"
      unfolding checkAddress_def a1 extractValueType.simps by simp
  qed
qed
```

```
lemmas exprTypeconInduct = exprTypeconInduct_aux[rule_format]

end
end
```

15 Supporting lemmas for statement preservation (TypeSafe_Statement_Support)

```
theory TypeSafe_Statement_Support
  imports TypeSafe_Expressions
begin

context typesafe_base
begin
```

15.1 Stack updates and value conversion

```
lemma diffTypeDiffLoc:
  assumes "st' = st(|Stack := updateStore l v (Stack st), Gas:=g|)"
    and "v' ≠ v"
    and "accessStore l1 (Stack st') = Some v'"
  shows "l1 ≠ l" using assms notNoneUpdate by fastforce

lemma stackSingleUpdate:
  assumes "st' = st(|Stack := updateStore l v (Stack st), Gas:=g|)"
  shows "∀nl ≠l. accessStore nl (Stack st) = accessStore nl (Stack st')"
proof (intros)
  fix nl
  assume a1:"nl ≠ l"
  then have "accessStore nl (updateStore l v (Stack st)) = accessStore nl (Stack st)"
    unfolding updateStore_def by (simp add: accessStore_def)
  then show "accessStore nl (Stack st) = accessStore nl (Stack st')" using assms by simp
qed

lemma updateOneType:
  assumes "st' = st(|Stack := updateStore l v (Stack st), Gas:=g|)"
    and "v' ≠ v"
  shows "accessStore l1 (Stack st') = Some v' ⟹ accessStore l1 (Stack st) = Some v'"
proof -
  assume a1:"accessStore l1 (Stack st') = Some v'"
  then have "l1 ≠ l" using diffTypeDiffLoc[of st' g l v st v' l1] assms by simp
  then show "accessStore l1 (Stack st) = Some v'" using stackSingleUpdate[of st' g l v st] assms a1 by
simp
qed


lemma typeSafeLocExists:
  assumes "TypeSafe ev (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "(t, l) |∈| fmran (Denvalue ev)"
    and "l = Stackloc loc"
    and "unique_locations (Denvalue ev)"
    and "balanceTypes (Accounts st)"
  shows "∃val. accessStore loc (Stack st) = Some(val)"
proof -
  have a10:"TypeSafe ev (Accounts st) (Stack st) (Memory st) (Storage st) cd = (unique_locations
(Denvalue ev) ∧
     balanceTypes (Accounts st) ∧
     (∀t l. (t, l) |∈| fmran (Denvalue ev) ⟶
            (denvalueCon (Stack st) (Memory st) (Storage st (Address ev)) cd t l)))"
    using assms TypeSafe_def typeCompat_def by simp
  then show ?thesis
  proof(cases "accessStore loc (Stack st)")
    case None
```

```
    then have "False" using a10 assms by force
    then show ?thesis by simp
  next
    case (Some a)
    then show ?thesis using a10 assms by simp
  qed
qed

lemma typeConConvert:
  assumes "local.expr ex env cd (st) g' = Normal ((KValue v, Value t2), g)"
    and "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "convert t2 t' v = Some (v)"
    and "fullyInitialised env (Accounts st) (Stack st)"
  shows "typeCon t' v"
proof -
  have "Memory (st) = Memory st" by simp
  moreover have "Storage (st) = Storage st" by simp
  moreover have "Accounts (st) = Accounts st" by simp
  moreover have "Stack st = Stack (st)" by simp
  ultimately have "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd" using assms(2)
by metis
  then have a10:"typeCon t2 v" using exprTypeconInduct(3) assms(1,4) by force
  then show "typeCon t' v" using a10 assms(3) typeSafeConvert by simp
qed
```

15.2 Locations returned by left-hand expressions

If storage is `SCon` and `local.lexp` returns a location in it, the types agree.

```
lemma lexpStorage:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "lexp lv env cd (st⦇Gas := g⦈) g = Normal ((LStoreloc locChanged, type.Storage (STValue
t')),g')"
    and "fullyInitialised env (Accounts st) (Stack st)"

obtains (1)
  "((type.Storage (STValue t'), Storeloc locChanged) |∈| fmran (Denvalue env))"
| (2) l t
where "((type.Storage t, Storeloc l) |∈| fmran (Denvalue env))"
  and "(TypedStoSubpref locChanged l t ∧ CompStoType t (STValue t') l locChanged)"

| (3) l' t l
where "((type.Storage t, Stackloc l) |∈| fmran (Denvalue env))"
  and "accessStore l (Stack st) = Some (KStoptr l')"
  and "(TypedStoSubpref locChanged l' t ∧ CompStoType t (STValue t') l' locChanged)"
proof (cases lv)
  case (Id x1)
  then have "lexp (Id x1) env cd (st⦇Gas := g⦈) g = Normal ((LStoreloc locChanged, type.Storage
(STValue t')), g')" using assms(2) by simp
  then have "Denvalue env $$ x1 = Some (type.Storage (STValue t'), Storeloc locChanged) "
    using Id lexp.simps(1)[of x1 env cd "(st⦇Gas := g⦈)" g] by (auto split:option.splits
denvalue.splits)
  then show ?thesis by (simp add: "1" fmranI)
next
  case (Ref x21 x22)
  then have *: "lexp (Ref x21 x22) env cd (st⦇Gas := g⦈) g = Normal ((LStoreloc locChanged,
type.Storage (STValue t')), g')" using assms(2) by simp
  then show ?thesis
  proof(cases "Denvalue env $$ x21")
    case None
    then show ?thesis using * lexp.simps(2) Ref by simp
  next
    case (Some a)
    then obtain tp loc where adef:"a = (type.Storage tp, loc)" using * lexp.simps(2) Ref by (cases a;
```

```
(auto split: result.splits type.splits denvalue.splits option.splits stackvalue.splits))
    then have b6:"(type.Storage tp, loc) |∈| fmran (Denvalue env)" using Some by (simp add: fmranI)

    then show ?thesis
    proof(cases loc)
      case (Stackloc x1)
      then show ?thesis
      proof(cases "accessStore x1 (Stack st)")
        case None
        then show ?thesis using * Ref Stackloc lexp.simps(2)[of x21 x22 env cd "(st(|Gas := g|))" g]
Some adef by simp
      next
        case some2:(Some a)
        then obtain x4 where x4def:"a = KStoptr x4" using * Ref Stackloc lexp.simps(2)[of x21 x22 env
cd "(st(|Gas := g|))" g] Some adef some2 by (cases a; auto)
        then obtain l'' t'' where b20:"ssel tp x4 x22 env cd (st(|Gas := g|)) g = Normal ((l'', t''),
g')"
          using Stackloc Some Ref * lexp.simps(2) adef some2 by (auto split: result.splits type.splits)
        have b10:"SCon tp x4 (Storage st (Address env))" using assms b6 some2 Stackloc x4def unfolding
TypeSafe_def typeCompat_def by fastforce
        then have b30:"locChanged = l'' ∧ (STValue t') = t''" using * lexp.simps(2) Ref adef Stackloc
b20 some2 x4def Some Ref b6 by simp

        then show ?thesis
        proof(cases x22)
          case Nil
          then have b25:"tp = t'' ∧ x4 = l''" using b20 ssel.simps(1) by simp
          then have b30:"locChanged = l'' ∧ (STValue t') = t''" using * lexp.simps(2) Ref adef
Stackloc some2 x4def b20 Some Ref b6 by simp
          then show ?thesis using b6 3 b25 Stackloc some2 x4def by auto
        next
          case (Cons a list)
          then have b10:"(CompStoType tp t'' x4 l'')"
            using exprTypeconInduct(2)[of tp x4 x22 env cd "(st(|Gas := g|))" g l'' t'' g'] assms
            using b10 b20 some2 x4def unfolding fullyInitialised_def by simp
          have b30:"l'' = locChanged ∧ (STValue t') = t''" using * lexp.simps(2) Ref adef Stackloc
some2 x4def b20 Some Ref b6 by simp
          then have "CompStoType tp (STValue t') x4 l''" using b10 by simp
          then show ?thesis using * lexp.simps(2) Ref adef Stackloc some2 x4def b20 Some Ref b6 b10
b30 3
            using CompStoType_imps_TypedStoSubpref by blast
        qed
      qed
    next
      case (Storeloc x2)
      then have b10:"SCon tp x2 (Storage st (Address env))" using assms b6 unfolding TypeSafe_def
typeCompat_def by force
      then obtain l'' t'' where b20:"ssel tp x2 x22 env cd (st(|Gas := g|)) g = Normal ((l'', t''), g')"
        using Storeloc Some Ref * lexp.simps(2) adef by (auto split: result.splits type.splits)
      then show ?thesis
      proof(cases x22)
        case Nil
        then have b25:"tp = t'' ∧ x2 = l''" using b20 ssel.simps(1) by simp
        then have b30:"locChanged = l'' ∧ (STValue t') = t''" using * lexp.simps(2) Ref adef Storeloc
b20 Some Ref b6 by simp
        then show ?thesis using b6 1 adef Some Ref * b25 b20 by (simp add: Storeloc)
      next
        case (Cons a list)
        then have b10:"CompStoType tp t'' x2 l''"
          using exprTypeconInduct(2)[of tp x2 x22 env cd "(st(|Gas := g|))" g l'' t'' g'] assms
          using b10 b20 Storeloc unfolding fullyInitialised_def by simp
        have b30:"l'' = locChanged ∧ (STValue t') = t''" using * lexp.simps(2) Ref adef Storeloc b20
Some Ref b6 by simp
        then have "CompStoType tp (STValue t') x2 l''" using b10 by simp
```

```
        then show ?thesis using b30 b6 2 adef Some Ref * b10 b20 Storeloc
          using CompStoType_imps_TypedStoSubpref by blast
      qed
    qed
  qed
qed
```

15.3 Storage variables of fully initialised environments

```
lemma typeSafe_storeloc_scon_from_denvalue:
  assumes ts: "TypeSafe env acc sk mem sto cd"
    and inDen: "(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
  shows "SCon tp i (sto (Address env))"
  using sameStoLocTSafe[OF ts] inDen by blast

lemma fi_contract_var_to_denvalue_storeloc:
  assumes fi: "fullyInitialised env acc sk"
    and addrEq: "Address e = Address env"
    and typedE: "Type (acc (Address e)) = Some (atype.Contract (Contract e))"
    and epLookup: "ep $$ Contract e = Some (ct, dud)"
    and ctVar: "ct $$ i = Some (Var tp)"
  shows "Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
proof -
  obtain c ct_fi dud_fi where
      fiType: "Type (acc (Address env)) = Some (atype.Contract c)"
    and fiCon: "Contract env = c"
    and fiEp: "ep $$ c = Some (ct_fi, dud_fi)"
    and fiMap: "(∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v,
Storeloc id)))"
    and fiLoc: "(∀ id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
    and fiPtr: "(∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue env) ∧ accessStore l sk = Some (KStoptr
p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t l'
p))"
    using fi unfolding fullyInitialised_def by blast
  have cEq: "Contract e = Contract env"
  proof -
    have "Type (acc (Address env)) = Some (atype.Contract (Contract e))"
      using typedE addrEq by simp
    moreover have "Type (acc (Address env)) = Some (atype.Contract (Contract env))"
      using fiType fiCon by simp
    ultimately show ?thesis by simp
  qed
  have epEnv: "ep $$ Contract env = Some (ct, dud)"
    using epLookup cEq by simp
  have epEnvFi: "ep $$ Contract env = Some (ct_fi, dud_fi)"
    using fiEp fiCon by simp
  have ctEq: "ct = ct_fi"
    using epEnv epEnvFi by auto
  have "ct_fi $$ i = Some (Var tp)"
    using ctVar ctEq by simp
  then show ?thesis
    using fiMap by blast
qed

lemma fiPtr_parent_from_fullyInitialised:
  assumes fi: "fullyInitialised env acc sk"
    and inDen: "(type.Storage t, Stackloc l) |∈| fmran (Denvalue env)"
    and ptr: "accessStore l sk = Some (KStoptr p)"
  shows "∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t l' p"
  using fi inDen ptr unfolding fullyInitialised_def by blast

lemma fi_denvalue_storeloc_to_contract_var:
```

```
  assumes fi: "fullyInitialised env acc sk"
    and epLookup: "ep $$ Contract env = Some (ct, dud)"
    and inDen: "(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
  shows "ct $$ i = Some (Var tp)"
proof -
  obtain c ct_fi dud_fi where
      fiType: "Type (acc (Address env)) = Some (atype.Contract c)"
    and fiCon: "Contract env = c"
    and fiEp: "ep $$ c = Some (ct_fi, dud_fi)"
    and fiMap: "(∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v,
Storeloc id)))"
    and fiLoc: "(∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
    and fiPtr: "(∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue env) ∧ accessStore l sk = Some (KStoptr
p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t l'
p))"
    using fi unfolding fullyInitialised_def by blast
  have epEnvFi: "ep $$ Contract env = Some (ct_fi, dud_fi)"
    using fiEp fiCon by simp
  have ctEq: "ct = ct_fi"
    using epLookup epEnvFi by auto
  obtain id where idDef: "Denvalue env $$ id = Some (type.Storage tp, Storeloc i)"
    using inDen by blast
  then have "id = i" using fiLoc by blast
  then have denI: "Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
    using idDef by simp
  then have "ct_fi $$ i = Some (Var tp)"
    using fiMap by blast
  then show ?thesis using ctEq by simp
qed

lemma safeContract_field_scon_from_typeCompat:
  assumes ts: "TypeSafe env acc sk mem sto cd"
    and fi: "fullyInitialised env acc sk"
    and addrEq: "Address e = Address env"
    and typedE: "Type (acc (Address e)) = Some (atype.Contract (Contract e))"
    and epLookup: "ep $$ Contract e = Some (ct, dud)"
    and ctVar: "ct $$ i = Some (Var tp)"
  shows "SCon tp i (sto (Address env))"
proof -
  have denLookup: "Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
    using fi_contract_var_to_denvalue_storeloc[OF fi addrEq typedE epLookup ctVar] .
  have inDen: "(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
    using denLookup by (simp add: fmranI)
  show ?thesis
    using typeSafe_storeloc_scon_from_denvalue[OF ts inDen] .
qed

lemma safeContract_other_address_preserved:
  assumes sc: "safeContract acc sto"
    and stoEq: "∀a. a ≠ a0 ⟶ sto' a = sto a"
    and addNe: "Address e ≠ a0"
    and typedE: "Type (acc (Address (e::environment))) = Some (atype.Contract (Contract e))"
    and epLookup: "ep $$ Contract e = Some (ct, dud)"
    and ctVar: "ct $$ i = Some (Var tp)"
  shows "SCon tp i (sto' (Address e))"
proof -
  have "SCon tp i (sto (Address e))"
    using sc typedE epLookup ctVar unfolding safeContract_def by blast
  moreover have "sto' (Address e) = sto (Address e)"
    using stoEq addNe by blast
  ultimately show ?thesis by simp
qed
```

```
lemma copy_dest_scon_from_rel:
  assumes copyStep: "copy p l x t (Storage st (Address env)) = Some s"
    and tEq: "t' = STArray x t"
    and rootSCon: "SCon (STArray x t) l (Storage st (Address env))"
    and srcSCon: "SCon (STArray x t) p (Storage st (Address env))"
    and denSub: "(if TypedStoSubpref stloc lsrc tsrc then CompStoType tsrc tp'' lsrc stloc
                 else if TypedStoSubpref lsrc stloc tp'' then CompStoType tp'' tsrc stloc lsrc else
True)"
    and paInf: "(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
                ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
    and rel: "CompStoType tsrc (STArray x t) lsrc l"
  shows "SCon (STArray x t) l s"
proof (cases "p = l")
  case pIsL: True
  then show ?thesis
    using copy_same_SCon copyStep tEq rootSCon by (metis copy_def)
next
  case pNotl: False
  then show ?thesis
  proof (cases "p = stloc")
    case True
    then have j1:"(if TypedStoSubpref p lsrc tsrc then CompStoType tsrc (STArray x t) lsrc p
                   else if TypedStoSubpref lsrc p (STArray x t) then CompStoType (STArray x t) tsrc p
lsrc else True)"
      using denSub paInf by fastforce
    then have j2:"¬TypedStoSubpref p l (STArray x t) ∧ ¬TypedStoSubpref l p (STArray x t)"
      using comp_stotype_same_type_same_loc j1 pNotl rel
      by (metis CompStoType_imps_TypedStoSubpref CompStoType_same_type_same_depth CompStoType_trns
NotReachablePrnt_imps_notReachableChild NotRelatedPrnt_imps_notRelatedChild
          same_depth_imp_neg_TypedStoSubpref)
    moreover have "∀ subL. TypedStoSubpref subL l (STArray x t) ⟶ ¬ TypedStoSubpref subL p (STArray
x t)"
      using NotRelatedPrnt_imps_notRelatedChild[of p l "(STArray x t)" "(STArray x t)"] j2 by blast
    moreover have "∀ subL. TypedStoSubpref subL p (STArray x t) ⟶ ¬ TypedStoSubpref subL l (STArray
x t)"
      using j2 NotRelatedPrnt_imps_notRelatedChild by blast
    ultimately show ?thesis
      using copy_notSub_SCon copyStep tEq rootSCon srcSCon by (metis (lifting) copy_def)
  next
    case False
    then have j1:"CompStoType tp'' (STArray x t) stloc p" using paInf by simp
    show ?thesis
    proof (cases "TypedStoSubpref stloc lsrc tsrc")
      case True
      then have "CompStoType tsrc tp'' lsrc stloc" using denSub by simp
      then have j5:"CompStoType tsrc (STArray x t) lsrc p"
        using CompStoType_trns j1 False by blast
      have j10:"CompStoType tsrc (STArray x t) lsrc l" using rel .
      then have j20:"lsrc ≠ l ∧ lsrc ≠ p" using pNotl j5 False
        using CompStoType_sameLoc_sameType comp_stotype_same_type_same_loc by blast
      then have j2:"¬TypedStoSubpref p l (STArray x t) ∧ ¬TypedStoSubpref l p (STArray x t)"
        using CompStoType_same_type_same_depth j5 j10 pNotl same_depth_imp_neg_TypedStoSubpref by blast
      moreover have "∀ subL. TypedStoSubpref subL l (STArray x t) ⟶ ¬ TypedStoSubpref subL p
(STArray x t)"
        using NotRelatedPrnt_imps_notRelatedChild[of p l "(STArray x t)" "(STArray x t)"] j2 by blast
      moreover have "∀ subL. TypedStoSubpref subL p (STArray x t) ⟶ ¬ TypedStoSubpref subL l
(STArray x t)"
        using j2 NotRelatedPrnt_imps_notRelatedChild by blast
      ultimately show ?thesis
        using copy_notSub_SCon copyStep tEq rootSCon srcSCon by (metis (lifting) copy_def)
    next
      case f2: False
```

```
      then show ?thesis
      proof (cases "TypedStoSubpref lsrc stloc tp''")
        case True
        then have j5:"CompStoType tp'' tsrc stloc lsrc" using denSub f2 by auto
        have j10:"CompStoType tp'' (STArray x t) stloc p" using j1 by simp
        have j11:"CompStoType tp'' (STArray x t) stloc l" using CompStoType_trns j5 rel by blast
        then have j2:"¬TypedStoSubpref p l (STArray x t) ∧ ¬TypedStoSubpref l p (STArray x t)"
          using CompStoType_same_type_same_depth j10 pNotl same_depth_imp_neg_TypedStoSubpref by blast
        moreover have "∀subL. TypedStoSubpref subL l (STArray x t) ⟶ ¬ TypedStoSubpref subL p
(STArray x t)"
          using NotRelatedPrnt_imps_notRelatedChild[of p l "(STArray x t)" "(STArray x t)"] j2 by blast
        moreover have "∀subL. TypedStoSubpref subL p (STArray x t) ⟶ ¬ TypedStoSubpref subL l
(STArray x t)"
          using j2 NotRelatedPrnt_imps_notRelatedChild by blast
        ultimately show ?thesis
          using copy_notSub_SCon copyStep tEq rootSCon srcSCon by (metis (lifting) copy_def)
      next
        case False
        then have j2:"¬TypedStoSubpref p l (STArray x t) ∧ ¬TypedStoSubpref l p (STArray x t)"
          using CompStoType_imps_TypedStoSubpref NotRelatedPrnt_imps_notRelatedChild f2 j1 rel
          by (meson NotReachablePrnt_imps_notReachableChild)
        moreover have "∀subL. TypedStoSubpref subL l (STArray x t) ⟶ ¬ TypedStoSubpref subL p
(STArray x t)"
          using NotRelatedPrnt_imps_notRelatedChild[of p l "(STArray x t)" "(STArray x t)"] j2 by blast
        moreover have "∀subL. TypedStoSubpref subL p (STArray x t) ⟶ ¬ TypedStoSubpref subL l
(STArray x t)"
          using j2 NotRelatedPrnt_imps_notRelatedChild by blast
        ultimately show ?thesis
          using copy_notSub_SCon copyStep tEq rootSCon srcSCon by (metis (lifting) copy_def)
      qed
    qed
  qed
qed
```

15.4 Memory selection

```
lemma originalMConStillMCon:
  assumes "MCon struct (Memory st) x3"
    and "(∀tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Memory st) = None)"
    and "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (Memory st) =
accessStore locs (Memory st')"
    and "∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) t' ⟶ accessStore
        locs (snd (allocate (Memory st))) = accessStore locs (Memory st')"
  shows " MCon struct (Memory st') x3" using assms(1)
proof(induction struct arbitrary: x3)
  case (MTArray x1 struct)
  have mcexp:"∀i<x1.
(case accessStore (hash x3 (ShowL_nat i)) (Memory st) of None ⇒ False
| Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct (Memory st)
(hash x3 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct (Memory st) loc2 | MTValue
Types ⇒ False))"
    using MTArray(2) unfolding MCon.simps by simp

  have "∀i<x1.
(case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
| Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct (Memory st')
(hash x3 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct (Memory st') loc2 | MTValue
Types ⇒ False))"
  proof(intros)
    fix i assume h1:"i<x1"
    show "case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
```

```
| Some (MValue val) ⇒ (case struct of MTArray l a ⇒ False | MTValue typ ⇒ MCon struct (Memory st')
(hash x3 (ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case struct of MTArray len' arr' ⇒ MCon struct (Memory st') loc2 | MTValue
Types ⇒ False)"
    proof(cases "struct")
      case mta1:(MTArray x11 x12)
      then obtain v where vdef: "accessStore (hash x3 (ShowL_nat i)) (Memory st) = Some (MPointer v)"
using h1 mcexp
        using MTArray.prems(1) MConArrayPointers by blast
      then have vdef2:"accessStore (hash x3 (ShowL_nat i)) (Memory st') = Some (MPointer v)" using
assms(2) assms(3,4)
        by (metis nat_le_linear option.distinct(1))
      have "MCon struct (Memory st) v" using vdef mcexp h1 mta1 by fastforce
      then have "MCon struct (Memory st') v" using MTArray.IH[of v]
        by simp
      then show ?thesis using mta1 h1 vdef vdef2 by simp
    next
      case (MTValue x2)
      then obtain v where vdef: "accessStore (hash x3 (ShowL_nat i)) (Memory st) = Some (MValue v)"
using h1 mcexp
        using MCon_sub_MTVal_imps_val MTArray.prems(1) by presburger
      then have vdef2:"accessStore (hash x3 (ShowL_nat i)) (Memory st') = Some (MValue v)" using assms
        by (metis nat_le_linear option.distinct(1))
      have "MCon struct (Memory st) (hash x3 (ShowL_nat i))" using vdef mcexp h1 MTValue by auto
      then show ?thesis using MTValue vdef vdef2 h1 by auto
    qed
  qed

  moreover have "x3 ≠ (ShowL_nat (Toploc (Memory st)))"
    using MCon_imps_Some MTArray.prems LSubPrefL2_def assms(2) not_None_eq by blast
  moreover have " ¬ TypedMemSubPref x3 (ShowL_nat (Toploc (Memory st))) t'" using assms
    by (metis (no_types, opaque_lifting) MCon_imps_Some MTArray.prems LSubPrefL2_def
Not_Sub_More_Specific le_refl neg_MemLSubPrefL2_imps_TypedMemSubPref not_Some_eq)
  moreover have "((∃p. accessStore x3 (Memory st') = Some (MPointer p)) ∨ accessStore x3 (Memory st')
= None)"
    using allocateSameAccess MTArray.prems calculation assms
    by (metis MCon.simps(2))

  ultimately show ?case unfolding MCon.simps
    by (metis MTArray.prems(1) MCon.simps(2))
next
  case (MTValue x')
  then show ?case using assms
    by (metis Option.option.simps(4) le_eq_less_or_eq MCon.simps(1))
qed


lemma mselReturnAlwaysHash:
  assumes"msel True t l'' x22 env cd st g = Normal ((l, t'), g')"
    and "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "MCon t (Memory st) l''"
    and "fullyInitialised env (Accounts st) (Stack st)"
  shows "∃prnt i len arr len' arr'. t= MTArray len arr ∧ l = hash prnt (ShowL_nat i) ∧ arr' = t'
      ∧ i < len' ∧ MCon (MTArray len' arr') (Memory st) prnt
                        ∧ (prnt = l'' ∧ len = len' ∧ arr' = arr ∨ CompMemType (Memory st) len arr
(MTArray len' arr') l'' prnt) " using assms(1,3)
proof(induction "x22" arbitrary: t l'' g)
  case Nil
  then show ?case using msel.simps by simp
next
  case (Cons a x22)
  show ?case using Cons(2)
  proof(cases rule: msel.elims )
    case (1 _ _ _ _ _ g)
```

```
    then show ?thesis by blast
  next
    case (2 _ _ _ _ _ _ g)
    then show ?thesis by auto
  next
    case (3 al t loc x env cd st g)
    then obtain kv b g4' where a20: "local.expr x env cd st g = Normal ((KValue kv, Value (TUInt b)),
g4')"
      and a30: "less (TUInt b) (TUInt b256) kv (ShowL_int (int al)) = Some ((ShowL_bool True), TBool)"
      using msel.simps(3) by (simp split: result.split_asm prod.split_asm types.split_asm
type.split_asm if_split_asm stackvalue.split_asm option.split_asm memoryvalue.split_asm)
    then have b2:"(hash loc kv, t) = (l, t')" using 3 by simp
    have a50: "checkUInt b kv" using exprTypeconInduct(3) 3(1) assms(2) a20 typeCon.simps(2)[of b
"extractValueType (KValue kv)"] extractValueType.simps(1)[of kv]
      using "3"(4,5,6) using assms(4) by force
    then have a60:"ReadL_int(kv) < int al " using a20 a30 less_def plift.simps(2)[of "(<)" b b256 kv
       "(ShowL_int (int al))"] Read_ShowL_id[of "(int al)"] unfolding createBool_def
       ShowL_bool_def by (simp split:if_split_asm)
    then have a62:"0≤ReadL_int(kv)" and a65:"(ShowL_int (ReadL_int kv)) = kv" using a50 checkUInt_def[of
b kv] by simp+
    have a64:"(ShowL_nat (nat (ReadL_int kv))) = kv" using ShowLnat_ReadLint_inverse a62 a50 unfolding
checkUInt_def by simp
    then have a70:"(nat (ReadL_int kv)) ∈ {0..al-1}" using a60 a62 by simp
    have "loc = l''" using 3 by simp
    moreover have "(nat (ReadL_int kv)) < al" using a70
      by (simp add: a60 a62 nat_less_iff)
    ultimately show ?thesis using b2 a64 a70 "3"(1) prod.inject
      by (metis Cons.prems(2))

  next
    case (4 al t'' loc x y ys env cd st g'')
    then obtain kv b g4' ptr where a20: "local.expr x env cd st g'' = Normal ((KValue kv, Value (TUInt
b)), g4')"
      and a30: "less (TUInt b) (TUInt b256) kv (ShowL_int (int al)) = Some ((ShowL_bool True), TBool)"
      and a40:"accessStore (hash loc kv) (if True then Memory st else cd) = Some (MPointer ptr)"
      using msel.simps(3) by (simp split: result.split_asm prod.split_asm types.split_asm
type.split_asm if_split_asm stackvalue.split_asm option.split_asm memoryvalue.split_asm)
    have a50: "checkUInt b kv" using exprTypeconInduct(3) 4(1) assms(2) a20 typeCon.simps(2)[of b
"extractValueType (KValue kv)"] extractValueType.simps(1)[of kv]
      using "4"(4,5,6) using assms(4) by force
    then have a60:"ReadL_int(kv) < int al " using a20 a30 less_def plift.simps(2)[of "(<)" b b256 kv
       "(ShowL_int (int al))"] Read_ShowL_id[of "(int al)"] unfolding createBool_def
       ShowL_bool_def by (simp split:if_split_asm)
    then have a62:"0≤ReadL_int(kv)" and a65:"(ShowL_int (ReadL_int kv)) = kv" using a50 checkUInt_def[of
b kv] by simp+
    have a64:"(ShowL_nat (nat (ReadL_int kv))) = kv" using ShowLnat_ReadLint_inverse a62 a50 unfolding
checkUInt_def by simp
    then have a70:"(nat (ReadL_int kv)) ∈ {0..al-1}" using a60 a62 by simp
    have tdef:"t = MTArray al t''" using 4 by simp
    then have mc:" MCon t'' (Memory st) ptr" using Cons.prems a40
      by (metis "4"(2,6) MCon_imps_sub_Mcon a60 a62 a64 nat_less_iff)

    then have mse:"msel True t'' ptr (y # ys) env cd st g4' = Normal ((l, t'), g')" using 4 a20 a30
a40 by simp
    then have "∃prnt i len arr len' arr'.
       t'' = MTArray len arr ∧
       l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = t' ∧ MCon (MTArray len' arr') (Memory st) prnt
           ∧ (prnt = ptr ∧ len = len' ∧ arr' = arr ∨ CompMemType (Memory st) len arr (MTArray len'
arr') ptr prnt )"
      using Cons.IH[of t'' ptr g4'] 4(4,5,6,7,3) a64 mc by blast
    then obtain prnt i len arr len' arr' where defs:"t'' = MTArray len arr ∧
       l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = t' ∧ MCon (MTArray len' arr') (Memory st) prnt
           ∧ (prnt = ptr ∧ len = len' ∧ arr' = arr ∨ CompMemType (Memory st) len arr (MTArray len'
arr') ptr prnt )" by
```

```
          blast

    have a6:"(case t' of MTArray l' ar' ⇒ ∃p. accessStore l (if True then Memory st else cd) = Some
(MPointer p)
                          ∧ CompMemType (if True then Memory st else cd) len arr t' ptr p
         | MTValue val ⇒ CompMemType (if True then Memory st else cd) len arr t' ptr l )"
      using exprTypeconInduct(1)[of True t'' ptr "(y # ys)" env cd st g4' l t' g'] mse mc assms(2) 4
defs
      using assms(4) unfolding fullyInitialised_def by simp

    then show ?thesis
    proof(cases "prnt = ptr")
      case True
      have notComp:"¬CompMemType (Memory st) len arr (MTArray len' arr') ptr prnt"
      proof
        assume in1:"CompMemType (Memory st) len arr (MTArray len' arr') ptr prnt"
        show False
        proof(cases "arr")
          case (MTArray x11 x12)
          then have "(∃i<len. ∃l. accessStore (hash ptr (ShowL_nat i)) (Memory st) = Some (MPointer l)
∧
                    (l = prnt ∧ MTArray x11 x12 = MTArray len' arr' ∨ CompMemType (Memory st) x11 x12
(MTArray len' arr') l prnt))"
            using in1 CompMemType.simps(2)[of "Memory st" len x11 x12 "(MTArray len' arr')" ptr prnt]
by auto
          moreover have " MCon (MTArray len (MTArray x11 x12)) (Memory st) ptr" using mc defs MTArray
by simp
          ultimately show ?thesis using True in1 mc defs
            by (metis BothMConImpsNotCompMemType)
        next
          case (MTValue x2)
          then show ?thesis using in1 True by simp
        qed
      qed

      then have lenIslen':"len = len'" using defs mc a6 notComp by simp
      then have arrIsarr':"arr' = arr" using defs mc a6 notComp by blast

      then have notSame:"ptr ≠ l''" using defs mc 4
        using Cons.prems(2) MConSubTypes CompMemJustType.simps(2) by blast
      have "CompMemType (Memory st) al (MTArray len arr) (MTArray len' arr') loc prnt "
        using CompMemType.simps(2)[of "Memory st" al len arr "(MTArray len' arr')" loc prnt] a40 True
defs lenIslen' arrIsarr'
        by (metis a60 a62 a64 nat_less_iff)
      moreover have "arr' = t'" using defs by simp
      ultimately show ?thesis using tdef True defs notSame
        using "4"(6,2) by blast
    next
      case False
      then have "CompMemType (Memory st) len arr (MTArray len' arr') ptr prnt" using defs by auto
      then have "CompMemType (Memory st) al (MTArray len arr) (MTArray len' arr') loc prnt"
        using CompMemType.simps(2)[of "Memory st" al len arr "(MTArray len' arr')" loc prnt] a40
        by (metis a60 a62 a64 nat_less_iff)
      then show ?thesis using tdef defs "4"(2,6) by blast
    qed
  qed
qed

lemma lexpIndexMem:
  assumes "lexp lv env cd st g = Normal ((LMemloc l, type.Memory t'), g')"
    and "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "fullyInitialised env (Accounts st) (Stack st)"

obtains (1) x21 x22 tp tParent l' l'' prnt len' arr' i
```

```
where "lv = Ref x21 x22"
  and "(tp, Stackloc l') |∈| fmran (Denvalue env)"
  and "accessStore l' (Stack st) = Some (KMemptr l'')"
  and "tp = type.Memory tParent"
  and "MCon tParent (Memory st) l'''"
  and "accessTypeStore l (Memory st) = Some (t')"
  and "(∃len arr. tParent = MTArray len arr ∧
      (case t' of MTArray l' ar' ⇒
          ∃p. accessStore l (Memory st) = Some (MPointer p) ∧ CompMemType (Memory st) len arr t' l'' p
      | MTValue val ⇒ CompMemType (Memory st) len arr t' l'' l )
∧ l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = t' ∧ MCon (MTArray len' arr') (Memory st) prnt
    ∧ (prnt = l'' ∧ len = len' ∧ arr' = arr ∨ CompMemType (Memory st) len arr (MTArray len' arr') l''
prnt))"
proof(cases lv)
  case (Id x1)
  then show ?thesis using assms(1) lexp.simps(1)[of x1] by (auto split:option.splits denvalue.splits)
next
  case (Ref x21 x22)
  then obtain tp l' where tpdef:"Denvalue env $$ x21 = Some (tp, Stackloc l')"
    using assms(1) lexp.simps(2)[of x21 x22 env cd "st"]
    by (auto split:option.splits denvalue.splits type.splits result.splits)
  then have parentIn:"( tp, Stackloc l') |∈| fmran (Denvalue env)" using fmranI by metis
  then obtain l'' where access:"accessStore l' (Stack st) = Some (KMemptr l'')"
    using Ref assms(1) lexp.simps(2)[of x21 x22 env cd "st"] tpdef
    by (auto split:option.splits denvalue.splits type.splits result.splits stackvalue.splits)
  then obtain t where tdef:"tp = type.Memory t" using access tpdef Ref assms(1) lexp.simps(2)[of x21
x22 env cd "st"]
    by (auto split:option.splits denvalue.splits type.splits result.splits stackvalue.splits)
  then have msel12:"msel True t l'' x22 env cd st g = Normal ((l, t'), g')" using access tpdef Ref
assms(1) lexp.simps(2)[of x21 x22 env cd "st"]
    by (auto split:option.splits denvalue.splits type.splits result.splits stackvalue.splits
prod.splits)
  have " MCon t (Memory st) l''"
    using access assms(2) parentIn sameMemTSafe tdef by blast
  then obtain prnt i len arr len' arr' where a1:"
     t = MTArray len arr ∧ l = hash prnt (ShowL_nat i) ∧ arr' = t' ∧ i < len' ∧ MCon (MTArray len'
arr') (Memory st) prnt
    ∧ (prnt = l'' ∧ len = len' ∧ arr' = arr ∨ CompMemType (Memory st) len arr (MTArray len' arr') l''
prnt)"
    using mselReturnAlwaysHash[OF msel12 assms(2)] assms(3) unfolding fullyInitialised_def by blast

  have mconP:"MCon t (Memory st) l'''" using tpdef access tdef assms(2)
    by (meson assms(2) fmranI sameMemTSafe)
  have "x22 ≠ []" using msel12 msel.simps by auto
  then have a6:"(∃len arr.
      t = MTArray len arr ∧
      (case t' of
       MTArray l' ar' ⇒
         ∃p. accessStore l (Memory st) = Some (MPointer p) ∧
             CompMemType (Memory st) len arr t' l'' p
       | MTValue val ⇒ CompMemType (Memory st) len arr t' l'' l ))"
    using exprTypeconInduct(1)[of True t l'' x22 env cd "st" g l t' g'] assms(2) mconP msel12 assms(3)
unfolding fullyInitialised_def
    by presburger

  have "((case t of MTArray len arr ⇒ ∀i<len. accessTypeStore (hash l'' (ShowL_nat i)) (Memory st) =
Some arr
       | MTValue val ⇒ accessTypeStore l'' (Memory st) = Some (MTValue val)))"
    using assms(2) unfolding TypeSafe_def denvalueTypeCorrectness_def
    using parentIn tdef access
    using a6 by fastforce
  then have a10:"(∀i<len. accessTypeStore (hash l'' (ShowL_nat i)) (Memory st) = Some arr)" using a1 by
simp
```

```
  have co:" (∀ locs tp.
      accessTypeStore locs (Memory st) = Some tp ⟶
      (case accessStore locs (Memory st) of None ⇒ False
       | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs
(Memory st) = Some tp
       | Some (MPointer p) ⇒ ∃ len arr. MCon tp (Memory st) p ∧ tp = MTArray len arr
                             ∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some arr)
                            ))"
    using assms(2) unfolding TypeSafe_def subPrefixStructuralConsistency_def by blast

  have "accessTypeStore l (Memory st) = Some t'"
  proof(cases "prnt = l'' ∧ len = len' ∧ arr' = arr ")
    case True
    then show ?thesis using a1 a10 by blast
  next
    case False
    then have cmp:"CompMemType (Memory st) len arr (MTArray len' arr') l'' prnt" using a1 by blast
    have accTT:"∀ i<len'. accessTypeStore (hash prnt (ShowL_nat i)) (Memory st) = Some arr'"
      using CompMemTypeSubIndexes[OF cmp ] a1 assms(2) unfolding TypeSafe_def
      using a10 mconP by argo
    then show ?thesis using a1 by simp
  qed

  then show ?thesis using msel12 1[OF Ref parentIn access tdef mconP, of prnt i len' arr' ] a1 a6 by
blast
qed

lemma lexpStorageG:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"

and "lexp lv env cd (st(|Gas := g|)) g = Normal ((LStoreloc locChanged, type.Storage t'),g')"
and "fullyInitialised env (Accounts st) (Stack st)"

obtains (1)
  "((type.Storage t', Storeloc locChanged) |∈| fmran (Denvalue env))"
| (2) l t
where "((type.Storage t, Storeloc l) |∈| fmran (Denvalue env))"
  and "(TypedStoSubpref locChanged l t ∧ CompStoType t t' l locChanged)"

| (3) l' t l
where "((type.Storage t, Stackloc l) |∈| fmran (Denvalue env))"
  and "accessStore l (Stack st) = Some (KStoptr l')"
  and "(TypedStoSubpref locChanged l' t ∧ CompStoType t t' l' locChanged)"
proof (cases lv)
  case (Id x1)
  then have "lexp (Id x1) env cd (st(|Gas := g|)) g = Normal ((LStoreloc locChanged, type.Storage (
t')), g')" using assms(2) by simp
  then have "Denvalue env $$ x1 = Some (type.Storage ( t'), Storeloc locChanged) "
    using Id lexp.simps(1)[of x1 env cd "(st(|Gas := g|))" g] by (auto split:option.splits
denvalue.splits)
  then show ?thesis by (simp add: "1" fmranI)
next
  case (Ref x21 x22)
  then have *: "lexp (Ref x21 x22) env cd (st(|Gas := g|)) g = Normal ((LStoreloc locChanged,
type.Storage ( t')), g')" using assms(2) by simp
  then show ?thesis
  proof(cases "Denvalue env $$ x21")
    case None
    then show ?thesis using * lexp.simps(2) Ref by simp
  next
    case (Some a)
    then obtain tp loc where adef:"a = (type.Storage tp, loc)" using * lexp.simps(2) Ref by (cases a;
(auto split: result.splits type.splits denvalue.splits option.splits stackvalue.splits))
    then have b6:"(type.Storage tp, loc) |∈| fmran (Denvalue env)" using Some by (simp add: fmranI)
```

```
    then show ?thesis
    proof(cases loc)
      case (Stackloc x1)
      then show ?thesis
      proof(cases "accessStore x1 (Stack st)")
        case None
        then show ?thesis using * Ref Stackloc lexp.simps(2)[of x21 x22 env cd "(st(|Gas := g|))" g]
Some adef by simp
      next
        case some2:(Some a)
        then obtain x4 where x4def:"a = KStoptr x4" using * Ref Stackloc lexp.simps(2)[of x21 x22 env
cd "(st(|Gas := g|))" g] Some adef some2 by (cases a; auto)
        then obtain l'' t'' where b20:"ssel tp x4 x22 env cd (st(|Gas := g|)) g = Normal ((l'', t''),
g')"
          using Stackloc Some Ref * lexp.simps(2) adef some2 by (auto split: result.splits type.splits)
        have b10:"SCon tp x4 (Storage st (Address env))" using assms b6 some2 Stackloc x4def unfolding
TypeSafe_def typeCompat_def by fastforce
        then have b30:"locChanged = l'' ∧ ( t') = t''" using * lexp.simps(2) Ref adef Stackloc b20
some2 x4def Some Ref b6 by simp

        then show ?thesis
        proof(cases x22)
          case Nil
          then have b25:"tp = t'' ∧ x4 = l''" using b20 ssel.simps(1) by simp
          then have b30:"locChanged = l'' ∧ ( t') = t''" using * lexp.simps(2) Ref adef Stackloc some2
x4def b20 Some Ref b6 by simp
          then show ?thesis using b6 3 b25 Stackloc some2 x4def
            using CompStoType_sameLocNdTyp TypedStoSubpref_sameLoc by auto
        next
          case (Cons a list)
          then have b10:"CompStoType tp t'' x4 l''"
            using exprTypeconInduct(2)[of tp x4 x22 env cd "(st(|Gas := g|))" g l'' t'' g'] assms
            unfolding fullyInitialised_def using b10 b20 some2 x4def by simp
          have b30:"l'' = locChanged ∧ (t') = t''" using * lexp.simps(2) Ref adef Stackloc some2 x4def
b20 Some Ref b6 by simp
          then have "CompStoType tp (t') x4 l''" using b10 by simp
          then show ?thesis using * lexp.simps(2) Ref adef Stackloc some2 x4def b20 Some Ref b6 b10
b30 3
            using CompStoType_imps_TypedStoSubpref by blast
        qed
      qed
    next
      case (Storeloc x2)
      then have b10:"SCon tp x2 (Storage st (Address env))" using assms b6 unfolding TypeSafe_def
typeCompat_def by force
      then obtain l'' t'' where b20:"ssel tp x2 x22 env cd (st(|Gas := g|)) g = Normal ((l'', t''), g')"
        using Storeloc Some Ref * lexp.simps(2) adef by (auto split: result.splits type.splits)
      then show ?thesis
      proof(cases x22)
        case Nil
        then have b25:"tp = t'' ∧ x2 = l''" using b20 ssel.simps(1) by simp
        then have b30:"locChanged = l'' ∧ (t') = t''" using * lexp.simps(2) Ref adef Storeloc b20 Some
Ref b6 by simp
        then show ?thesis using b6 1 adef Some Ref * b25 b20 by (simp add: Storeloc)
      next
        case (Cons a list)
        then have b10:"CompStoType tp t'' x2 l''"
          using exprTypeconInduct(2)[of tp x2 x22 env cd "(st(|Gas := g|))" g l'' t'' g'] b20 assms
          unfolding fullyInitialised_def using b10 b20 Storeloc by simp
        have b30:"l'' = locChanged ∧ (t') = t''" using * lexp.simps(2) Ref adef Storeloc b20 Some Ref
b6 by simp
        then have "CompStoType tp (t') x2 l''" using b10 by simp
        then show ?thesis using b30 b6 2 adef Some Ref * b10 b20 Storeloc
```

```
          using CompStoType_imps_TypedStoSubpref by blast
      qed
    qed
  qed
qed
```

15.5 The state invariant

StateInvariant is the conclusion of the main theorem. The resulting state is type-safe and fully initialised, and memory only grows: existing types, pointers and values are kept, unallocated locations below the old top stay unallocated, and the top location does not decrease.

```
definition allStoresSCon::"(address ⇒ storageT) ⇒ bool"
  where "allStoresSCon stoN = (∀ e ct t' l'. ep $$ Contract e = Some ct
∧ (type.Storage t', Storeloc l') |∈| fmran (Denvalue (e::environment)) ⟶ (SCon t' l' (stoN (Address
(e::environment)))))"


definition StateInvariant :: "environment ⇒ state ⇒ state ⇒ calldataT ⇒ bool" where
  "StateInvariant ev st_orig st_curr cd ≡
     TypeSafe ev (Accounts st_curr) (Stack st_curr) (Memory st_curr) (Storage st_curr) cd ∧
     fullyInitialised ev (Accounts st_curr) (Stack st_curr) ∧
     (∀ locs t. accessTypeStore locs (Memory st_orig) = Some t ⟶ accessTypeStore locs (Memory st_curr)
= Some t) ∧
(∀ locs v. accessStore locs (Memory st_orig) = Some (MPointer v) ⟶ (∃ v'. accessStore locs (Memory
st_curr) = Some (MPointer v'))) ∧
(∀ locs v. accessStore locs (Memory st_orig) = Some (MValue v) ⟶ (∃ v'. accessStore locs (Memory
st_curr) = Some (MValue v'))) ∧
(∀ i loc. i < Toploc (Memory st_orig) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
                      accessStore loc (Memory st_orig) = None ⟶ accessStore loc (Memory st_curr) =
None) ∧
  Toploc (Memory st_orig) ≤ Toploc (Memory st_curr)
    "
```

15.6 Memory-to-memory copies

```
lemma cpm2m_subPrefixPersist:
  assumes "subPrefixStructuralConsistency mo"
    and "accessStore (ShowL_nat (Toploc mo)) mo = None"
    and "SomeValSomeTyp mo"
    and sameTypeAccess:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mo)) ∨ locs = (ShowL_nat (Toploc
mo)) ⟶
                          accessTypeStore locs (mo) = accessTypeStore locs m"
    and lessThan:"lessThanTopLocs mo"
    and sameTypeAccess2:" ∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc mo)) (MTArray x t) ∨ locs =
        (ShowL_nat (Toploc mo)) ⟶
       accessTypeStore locs (mo) = accessTypeStore locs m"
    and tps:"∀ destl'.
     TypedMemSubPref destl' (ShowL_nat (Toploc mo)) (MTArray x t) ⟶
     (∃ stt. CompMemType m x t stt (ShowL_nat (Toploc mo)) destl' ∧
           (case stt of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m = Some parent_arr
            | MTValue pval ⇒ accessTypeStore destl' m = Some (MTValue pval)))"
    and MCondest:" MCon (MTArray x t) m (ShowL_nat (Toploc (mo)))"
    and selfPoint:"∀ l l'. TypedMemSubPref l (ShowL_nat (Toploc mo)) (MTArray x t) ∧ accessStore l m =
Some (MPointer l') ⟶ l' = l"
    and tps2:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc mo)) (ShowL_nat i)) m = Some t"
    and nonLocChanged:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc mo)) ∨ locs = (ShowL_nat (Toploc mo))
                 ⟶ accessStore locs (mo) = accessStore locs m"
    and a32:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc mo)) (MTArray x t) ⟶ accessStore locs
mo = accessStore locs m"
  shows "subPrefixStructuralConsistency (m)" unfolding subPrefixStructuralConsistency_def
proof intros
  fix locs tp
```

```
  assume in1:"accessTypeStore locs m = Some tp"
  show "case accessStore locs m of None ⇒ False
 | Some (MValue v) ⇒ ∃val. MCon tp m locs ∧ tp = MTValue val ∧ accessTypeStore locs m = Some tp
 | Some (MPointer p) ⇒
     ∃len arr.
        MCon tp m p ∧
        tp = MTArray len arr ∧
        (∀i<len. accessTypeStore (hash p (ShowL_nat i)) m = Some arr) "
  proof(cases " locs = (ShowL_nat (Toploc mo)) ")
    case True
    then have "accessStore locs mo = None"
      by (simp add: assms(2))
    then have "accessTypeStore locs mo = None"
      using assms(3) unfolding SomeValSomeTyp_def by force
    then have "accessTypeStore locs m = None" using sameTypeAccess True by simp
    then show ?thesis using in1 by simp
  next
    case notTop:False
    then show ?thesis
    proof(cases "LSubPrefL2 locs (ShowL_nat (Toploc mo))")
      case True
      then have "accessStore locs mo = None "
        using lessThan unfolding lessThanTopLocs_def by blast
      then have "accessTypeStore locs mo = None"
        using assms(3) unfolding SomeValSomeTyp_def by force
      then have "TypedMemSubPref locs (ShowL_nat (Toploc mo)) (MTArray x t)"
        using sameTypeAccess2 in1 by auto
      then obtain stt where sttDef:"( CompMemType m x t stt (ShowL_nat (Toploc mo)) locs ∧
        (case stt of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash locs
(ShowL_nat i)) m = Some parent_arr
         | MTValue pval ⇒ accessTypeStore locs m = Some (MTValue pval)))" using tps by blast
      then have mcLocs: "MCon stt m locs"
        using CompTypeRemainsMCon MCondest by blast
      then show ?thesis
      proof(cases stt)
        case (MTArray x11 x12)

        then have mi:"t = MTArray x11 x12 ∧ (∃i<x. accessStore (hash (ShowL_nat (Toploc mo)) (ShowL_nat
i)) m = Some (MPointer locs)) ∨
                  (∃midP subL subA i. CompMemType m x t (MTArray subL subA) (ShowL_nat (Toploc mo)) midP
∧
                    accessStore (hash midP (ShowL_nat i)) m = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
          using CompMemType_imps_Mid[of m x t x11 x12 "(ShowL_nat (Toploc mo))" locs] sttDef by simp

        then show ?thesis
        proof(cases "t = MTArray x11 x12 ∧ (∃i<x. accessStore (hash (ShowL_nat (Toploc mo)) (ShowL_nat
i)) m = Some (MPointer locs))")
          case True
          then obtain i where idef:"i<x ∧ accessStore (hash (ShowL_nat (Toploc mo)) (ShowL_nat i)) m =
Some (MPointer locs)"
            by blast
          then have isTop:"locs = (hash (ShowL_nat (Toploc mo)) (ShowL_nat i))"
            using selfPoint by auto
          then have tpDef:"t = tp" using tps2 in1 idef by simp
          have "accessStore locs m = Some (MPointer locs)"
            using MCondest idef isTop by blast
          moreover have "∃len arr.
                MCon tp m locs ∧
                tp = MTArray len arr ∧
                (∀i<len. accessTypeStore (hash locs (ShowL_nat i)) m = Some arr) ∧
                (∀v. accessStore locs m = Some (MPointer v) ⟶ accessTypeStore locs m = Some (MTArray
len arr))"
            using tpDef mcLocs True MTArray
```

```
          using in1 sttDef by fastforce
        ultimately show ?thesis using in1 MTArray mcLocs tps2 by auto
      next
        case False
        then have prefPtrs_imps_Pref:"∀ l. TypedMemSubPrefPtrs m x t (ShowL_nat (Toploc mo)) l
                                  ⟶ TypedMemSubPref l (ShowL_nat (Toploc mo)) (MTArray x t)"
          using selfPoint_imps_TypedMemSubPref[of " (ShowL_nat (Toploc mo))" x t m ] selfPoint by
blast

        then obtain midP subL subA i
          where mid: "(CompMemType m x t (MTArray subL subA) (ShowL_nat (Toploc mo)) midP ∧
                 accessStore (hash midP (ShowL_nat i)) m = Some (MPointer locs) ∧ i < subL ∧ subA =
MTArray x11 x12)"
          using mi False by auto
        then have locD:"locs = (hash midP (ShowL_nat i))" using selfPoint
          using CompMemType_imps_in_GetAllMemoryLocations_ptr MCondest memSet_selfPoint by blast
        then obtain st2 where st2D:"(CompMemType m x t st2 (ShowL_nat (Toploc mo)) midP ∧
       (case st2 of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash midP
(ShowL_nat i)) m = Some parent_arr
        | MTValue pval ⇒ accessTypeStore midP m = Some (MTValue pval)))" using mid tps
          using CompMemTypeSameLocsSameType CompMemType_imps_TypedMemSubPrefPtrs prefPtrs_imps_Pref
by blast
        then have "st2 = (MTArray subL subA)" using mid CompMemTypeSameLocsSameType MCondest by
blast
        then have " ∀ i<subL. accessTypeStore (hash midP (ShowL_nat i)) m = Some subA" using st2D by
simp
        then have tpDef:"MTArray x11 x12 = tp" using in1 locD mid by simp
        moreover have "∃ len arr.
                       MCon tp m locs ∧
                       tp = MTArray len arr ∧
                       (∀ i<len. accessTypeStore (hash locs (ShowL_nat i)) m = Some arr) "
          using tpDef mcLocs True MTArray in1 sttDef by auto
        ultimately show ?thesis using mid locD by simp
      qed
    next
      case (MTValue x2)
      then obtain v where isVal:"accessStore locs m = Some (MValue v)" using mcLocs
        using MConAccessSame.simps(1) SameMCon_imps_MConAccessSame by blast
      have "∃ val. MCon tp m locs ∧ tp = MTValue val ∧ accessTypeStore locs m = Some tp"
        using mcLocs in1 sttDef MTValue by simp
      then show ?thesis using isVal by simp
    qed
  next
    case False
    then have sameAccT:"accessTypeStore locs mo = accessTypeStore locs m"
      using sameTypeAccess by blast
    have sameAcc:"accessStore locs mo = accessStore locs m"
      using False nonLocChanged by metis
    obtain a where acc1:"accessStore locs mo = Some a"
      using assms(3) sameAccT unfolding TypeSafe_def SomeValSomeTyp_def using in1 by auto
    then have old:"(case accessStore locs mo of
    Some (MValue v) ⇒ ∃ val. MCon tp mo locs ∧ tp = MTValue val ∧ accessTypeStore locs mo = Some tp
  | Some (MPointer p) ⇒
     ∃ len arr.
        MCon tp mo p ∧
        tp = MTArray len arr ∧
        (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) mo = Some arr))"
      using assms(1) in1 sameAccT unfolding subPrefixStructuralConsistency_def by fastforce
    then show ?thesis
    proof(cases a)
      case (MValue x1)

      then show ?thesis using sameAccT acc1 old MValue a32 in1 MCon.simps sameAcc
        by (auto split:option.splits memoryvalue.splits)
```

```
      next
        case (MPointer x2)

        then have oo:"∃len arr.
          MCon tp mo x2 ∧
          tp = MTArray len arr ∧
          (∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) mo = Some arr) "
          using old acc1 by simp
        then have subs:"¬ LSubPrefL2 (ShowL_nat (Toploc mo)) x2 ∧¬ LSubPrefL2 x2 (ShowL_nat (Toploc mo))
"
          using assms unfolding TypeSafe_def
          by (meson AllPtrsNotTop_MCon TypeSafe_def le_refl)
        have "MCon tp mo x2" using oo by simp
        then have "MCon tp m x2"
          using lessThanTop_topChange_MCon subs nonLocChanged lessThan by simp

        then have "∃len arr.
     MCon tp m x2 ∧
     tp = MTArray len arr ∧
     (∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) m = Some arr) "
          using oo subs nonLocChanged by (simp add: Mutual_NonSub_SpecificNonSub sameTypeAccess)
        moreover have "accessStore locs m = Some (MPointer x2)"
          using MPointer sameAcc acc1 by simp
        ultimately show ?thesis by simp
      qed
    qed
  qed
qed

lemma cpm2m_singleLChange:
  assumes "MCon struct m x3"
  assumes structss:"(case struct of
       MTArray len arr ⇒
         (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) m = Some arr)
       | MTValue val ⇒ accessTypeStore x3 m = Some (MTValue val))"
    and "updateStore l (MPointer tloc) m = m'"
    and lOld2:"∃p. accessStore l m = Some (MPointer p)"
    and acc1:"accessTypeStore l (m) = Some (MTArray x t)"
    and MCondest2:"MCon (MTArray x t) m' (tloc)"
    and subPrefM:"subPrefixStructuralConsistency (m)"
    and NonChangeM'm:"∀locs. locs ≠ l ⟶ accessStore locs m = accessStore locs m'"
  shows "MCon struct m' x3"
  using assms(1,2)
proof(induction struct arbitrary:x3)
  case (MTArray x11 x12)
  have accessL:"accessStore l m' = Some (MPointer tloc)"
    using assms unfolding updateStore_def accessStore_def by auto
  have oldexp:"(∀i<x11.
          case accessStore (hash x3 (ShowL_nat i)) m of None ⇒ False
          | Some (MValue val) ⇒
              (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 m (hash x3 (ShowL_nat i)))
          | Some (MPointer loc2) ⇒
              (case x12 of MTArray len' arr' ⇒ MCon x12 m loc2 | MTValue val ⇒ False)) ∧
      (∃p. accessStore x3 m = Some (MPointer p) ∨ accessStore x3 m = None)"
    using MTArray.prems(1) unfolding MCon.simps by simp

  then have subs:"(∀i<x11. accessTypeStore (hash x3 (ShowL_nat i)) m = Some x12)"
    using structss MTArray by simp

  have "(∀i<x11.
          case accessStore (hash x3 (ShowL_nat i)) m' of None ⇒ False
          | Some (MValue val) ⇒
              (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 m' (hash x3 (ShowL_nat i)))
          | Some (MPointer loc2) ⇒
```

```
            (case x12 of MTArray len' arr' ⇒ MCon x12 m' loc2 | MTValue val ⇒ False))"
  proof(intros)
    fix i assume idef:"i<x11"
    then consider (val1) val typw where "accessStore (hash x3 (ShowL_nat i)) m = Some (MValue val)
                              ∧ x12 = MTValue typw
                              ∧ MCon x12 m (hash x3 (ShowL_nat i))"
      | (ptr1) ptr l1 a1 where "accessStore (hash x3 (ShowL_nat i)) m = Some (MPointer ptr)
                           ∧ x12 = MTArray l1 a1
                           ∧ MCon x12 m ptr"
      using oldexp
      by (metis MCon_imps_sub_Mcon MTArray.prems(1) mtypes.distinct(1) gr_implies_not0 mcon_accessStore
nat_neq_iff)
    then show "case accessStore (hash x3 (ShowL_nat i)) m' of None ⇒ False
| Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 m' (hash x3
(ShowL_nat i)))
| Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12 m' loc2 | MTValue val ⇒ False)"
    proof(cases)
      case val1
      then show ?thesis
      proof(cases "(hash x3 (ShowL_nat i)) = l")
        case True
        then show ?thesis using True lOld2 val1 by simp
      next
        case False
        then have sameACC:"accessStore (hash x3 (ShowL_nat i)) m = accessStore (hash x3 (ShowL_nat i))
m'"
          using assms unfolding updateStore_def accessStore_def by auto
        then have "MCon x12 m' (hash x3 (ShowL_nat i))"
          using val1 unfolding MCon.simps by auto
        then show ?thesis using val1 sameACC by simp
      qed
    next
      case ptr1
      then have "∀i<x11. accessTypeStore (hash x3 (ShowL_nat i)) m = Some x12" using subs by simp

      then show ?thesis
      proof(cases "(hash x3 (ShowL_nat i)) = l")
        case True
        then have "accessStore (hash x3 (ShowL_nat i)) m' = Some (MPointer (tloc))"
          using assms unfolding updateStore_def accessStore_def by auto

        moreover have "x12 = MTArray x t" using subs True idef MTArray.prems calculation accl by auto
        moreover have "MCon x12 m' (tloc)" using calculation MCondest2 by blast
        ultimately show ?thesis
          by (simp add: ptr1)
      next
        case False
        then have "case x12 of MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr (ShowL_nat i)) m =
Some arr)
                  | MTValue val ⇒ accessTypeStore ptr m = Some (MTValue val)"
          using subPrefM unfolding subPrefixStructuralConsistency_def using MTArray.prems
          using ptr1 idef by fastforce
        then have "MCon x12 m' ptr"
          using MTArray.IH[of ptr] ptr1 by blast
        then show ?thesis
          using False NonChangeM'm ptr1 by auto
      qed
    qed
  qed
  then show ?case using MTArray.prems(1) unfolding MCon.simps
    by (metis NonChangeM'm accessL)
next
  case (MTValue x2)
  then show ?case
```

```
  proof(cases "x3 = l")
    case True
    then show ?thesis using accl
      using MTValue.prems(2) by auto
  next
    case False
    then have "accessStore x3 m = accessStore x3 (m')"
      using assms unfolding updateStore_def accessStore_def by auto
    then show ?thesis using MTValue unfolding MCon.simps by simp
  qed
qed

end

end
```

16 Preservation for assignments (TypeSafe_Assign)

```
theory TypeSafe_Assign
  imports TypeSafe_Statement_Support
begin

context typesafe_base
begin
```

Each lemma covers one case of the parent's *assign* rule for *ASSIGN*, distinguished by the kinds of the assigned value and of the target location. Together they give the *ASSIGN* case of *TypeSafe_Statements*.

Assignment case 1: a value assigned to a stack variable.

```
lemma TypeSafe_Assign_1:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st⦇state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈) (state.Gas st
- costs (ASSIGN lv ex) env cd st) = Normal ((KValue v, Value t2), g)"
    and "lexp lv env cd (st⦇state.Gas := g⦈) g = Normal((LStackloc l2, Value t'),g')"
    and "convert t2 t' v = Some v'"
    and "st' = st⦇state.Gas := g', Stack := updateStore l2 (KValue v') (Stack st)⦈"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 1 = assms(4-7)
  show "StateInvariant env st st' cd" unfolding StateInvariant_def TypeSafe_def
  proof (intros)
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd" unfolding
typeCompat_def
    proof intros
      fix t l assume a10:"(t, l) |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t l"
      proof (split denvalue.split, intros)
        fix loc assume a20:"l = Stackloc loc"
        show "case accessStore loc (Stack st') of None ⇒ False
              | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
              | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
              | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc
| _ ⇒ False)
              | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
        proof (cases "loc = l2")
          case False
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
```

```
          then have "accessStore loc (Stack st) = None" using 1(4) False by (simp
add:stackSingleUpdate)
          then show False using a30 a20 a10 assms False 2(1) unfolding TypeSafe_def typeCompat_def
by force
        next
          fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
          then have a40:"accessStore loc (Stack st) = Some x2" using 1(4) False by (simp
add:stackSingleUpdate)
          then have a50:"(Memory st) = (Memory st')" using 1(4) by simp
          then have a60:"(Storage st) = (Storage st')" using 1(4) by simp
          show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
                | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
                | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc |
_ ⇒ False)
                | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
            using a10 a20 a30 a40 a50 a60 2(1) unfolding TypeSafe_def typeCompat_def by (cases x2;
cases t; force)
        qed
      next
        case sameLoc:True
        show ?thesis
        proof (split option.split, intros)
          assume a30:"accessStore loc (Stack st') = None"
          then show False using a20 a10 assms(1) sameLoc 1(4) notNoneUpdate[of st' g' loc "KValue
v'" ] by simp
        next
          fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
          show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
            | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
            | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
            | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
          proof(cases "x2")
            case (KValue x1)
            then have a40:"unique_locations (Denvalue env)" using 2(1) typeSafeUnique by simp
            then have "(Value t', l2) = (t, loc)" using a40 a20 1(2) lexpStackloc_imps_inDen a10
sameLoc
                uniqueLocs[of env "(t,l)" "(Value t', Stackloc l2)"] by simp
            then have True:"Value t' = t" by simp
            then have a40:"v' = v" using convertSame[of t2 t' v v'] 1(3) by simp
            moreover have "typeCon t' v" using a40 1 2(1) 2(3)
              by (simp add: typeConConvert)
            moreover have "x1 = v"
              using sameLoc a30 a40 KValue 1(4) notNoneUpdate[of st' g' l2 "(KValue x1)" ]
notNoneUpdate[of st' g' l2 "(KValue v')" ] by simp
            ultimately have "typeCon t' x1" by simp
            then show ?thesis using True KValue by auto
          next
            case (KCDptr x2)
            then show ?thesis using a30 "1"(4) by (simp add:sameLoc)
          next
            case (KMemptr x3)
            then show ?thesis using a30 "1"(4) by (simp add:sameLoc)
          next
            case (KStoptr x4)
            then show ?thesis using a30 "1"(4) by (simp add:sameLoc)
          qed
        qed
      qed

    next
      fix x2 assume a20:"l = Storeloc x2"
```

```
      then have "(Storage st' (Address env)) = (Storage st (Address env))" using 1(4) by simp
      then show "case t of type.Storage typ ⇒ SCon typ x2 (Storage st' (Address env)) | _ ⇒ False"
        using a10 a20 2(1) unfolding TypeSafe_def typeCompat_def by (cases t; force)
    qed
  qed
 next
  show "unique_locations (Denvalue env)" using 2(1) typeSafeUnique by auto
 next
  have "(Accounts st) = Accounts(st')" using 1(4) by simp
  then show "balanceTypes (Accounts st')" using balanceTypes_def balanceTypes_def 2(1)
typeSafeAccounts by simp
 next
  show " compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
  proof(intros)
    fix tp1 tp2 l1 l22 l1' l2' stl1 stl2
    assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
   (type.Storage tp2, l22) |∈| fmran (Denvalue env) ∧
   (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
   (l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l22 = Storeloc stl2)"
    then have 10:"accessStore l1' (Stack st) = Some (KStoptr stl1) ∨ l1 = Storeloc stl1" using 1(4)
2(1) updateOneType[of st' g' l2 "(KValue v')" st "(KStoptr stl1)" l1' ] by auto
    then have 20:"accessStore l2' (Stack st) = Some (KStoptr stl2) ∨ l22 = Storeloc stl2" using a1
1(4) 2(1) updateOneType[of st' g' l2 "(KValue v')" st "(KStoptr stl2)" l2' ] by auto
    have "Storage st' (Address env) = (Storage st (Address env))" using a1 1(4) 2(1) updateOneType[of
st' g' l2 "(KValue v')" st "(KStoptr stl2)" l2' ] by auto
    then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      using 2(1) a1 10 20 unfolding TypeSafe_def compPointers_def by blast
  qed
 next
  show "svalueTypes (Svalue env)" using svalueTypes_def typeSafeSvalue 2(1) by simp
 next
  have "(Storage st') = (Storage st)" using 1(4) by simp
  then show "safeContract (Accounts st') (Storage st')" using 2(1) 1(4) unfolding safeContract_def
TypeSafe_def
    by fastforce
 next
  have a10:"Toploc (Stack st') = Toploc (Stack st)" using 1(4) unfolding updateStore_def by simp
  then have "(Value t', Stackloc l2) |∈| fmran (Denvalue env)" using 2(1) 1(2)
lexpStackloc_imps_inDen by simp
  then have a20:"∃ val. accessStore l2 (Stack st) = Some val" using typeSafeLocExists 2(1)
TypeSafe_def by blast
  then have a30:"(∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st).
LSubPrefL2 loc (ShowL_nat tloc)))"
    using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have a40:"(∀ tloc loc. Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                 ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st').
LSubPrefL2 loc (ShowL_nat tloc)))"
    using a10 by simp
  show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
  proof intros
    fix tloc loc
    assume *:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
    then show "accessStore loc (Stack st') = None"
    proof(cases "loc = l2")
      case True
      then show ?thesis using * a10
        by (metis a20 a30 option.distinct(1))
    next
      case False
      then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 1(4) unfolding
```

```
updateStore_def accessStore_def by simp
        then show ?thesis using 2(1) a40 * a10 False a30 by simp
      qed
    next
      fix loc y
      assume *:" accessStore loc (Stack st') = Some y "
      show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)"
      proof(cases "loc = l2")
        case True
        then show ?thesis using *a10 a20 a30 by simp
      next
        case False
        then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 1(4) unfolding
updateStore_def accessStore_def by simp
        then show ?thesis using * 2(1) a40 by simp
      qed
    qed
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def using 1 by auto
  next
    show "lessThanTopLocs (Memory st')" using 2(1) unfolding TypeSafe_def using 1 by auto
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by auto
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using 1 by simp
  next
    have "Accounts st'= Accounts st" using 1 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 1(4) unfolding
fullyInitialised_def updateStore_def accessStore_def by auto
  next
    have cc0:"∀l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
      using 1(4) unfolding updateStore_def accessStore_def by auto
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"
      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
       MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      moreover have "Memory st = Memory st'" using 1(4) by simp
      ultimately show "case t of
     MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        by metis
    qed
  next
    show "subPrefixStructuralConsistency (Memory st')"
      using 2(1) 1(4) unfolding TypeSafe_def by simp
  next
    show "SomeValSomeTyp (Memory st') " using 2(1) unfolding TypeSafe_def using 1(4) by auto
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
      using 1 by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 1 by simp
  next
```

```
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 1 by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 1 by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 1 by simp
  qed
qed
```

Assignment case 2: a value assigned to a storage location.

```
lemma TypeSafe_Assign_2:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv ex) env cd st) = Normal ((KValue v, Value t2), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStoreloc locationChanged, type.Storage
(STValue t')),g')"
    and "convert t2 t' v = Some v'"
    and "st' = st(|state.Gas := g', Storage := (Storage st) (Address env := (fmupd locationChanged v'
(Storage st (Address env))))|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note subcase2 = assms(4-7)
  then have tpCon:"typeCon t2 (extractValueType (KValue v)) ∧ (∃xx. KValue v = KValue xx)"
    using exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd
st|))" "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KValue v" "Value t2" g]
    using 2(1) 2(3) unfolding fullyInitialised_def by force
  then have tcon:"typeCon t' v'" using typeSafeConvert[of t2 "(extractValueType (KValue v))" t' ]
subcase2(3)
    by (metis convertSame extractValueType.simps(1))
  have stacksSame:"Stack st = Stack st'" using subcase2(4) by simp
  have accessStoreSt:"Storage st' (Address env) $$ locationChanged = Some v'" using subcase2 by simp
  then have scont':"SCon (STValue t') locationChanged (Storage st' (Address env))"
    using SCon.simps(1)[of t' locationChanged "Storage st' (Address env)"] accessStorage_def tcon by
simp
  have nonLocChanged:"∀l. l ≠ locationChanged ⟶ Storage st' (Address env) $$ l = Storage st
(Address env) $$ l" using subcase2 by simp
  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "unique_locations (Denvalue env)" using 2(1) typeSafeUnique by simp
  next
    have "compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def using 2(1)
compPointers_def stacksSame typeSafeCompPointers by force

  next
    show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
    proof intros
      fix e ct dud i tp
      assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
               ep $$ Contract (e::environment) = Some (ct, dud) ∧
               ct $$ i = Some (Var tp)"
      have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp
      show "SCon tp i (Storage st' (Address e))"
      proof (cases "Address e = Address env")
        case False
        have stoEq:"∀a. a ≠ Address env ⟶ Storage st' a = Storage st a"
          using subcase2(4) by simp
```

```
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * subcase2(4) by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        show ?thesis
          using safeContract_other_address_preserved[OF scOld stoEq False typedOld epOld ctOld] .
      next
        case addrEq:True
        obtain c where **:"Type (Accounts st (Address env)) = Some (atype.Contract c) ∧ Contract env =
c"
          using 2(3) unfolding fullyInitialised_def using 2 by blast
        have cEq:"Contract e = Contract env"
          using * ** addrEq subcase2(4) by auto
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * subcase2(4) by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
          using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
        have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
          using denI by (simp add: fmranI)
        have oldSCon:"SCon tp i (Storage st (Address env))"
          using safeContract_field_scon_from_typeCompat[OF 2(1) 2(3) addrEq typedOld epOld ctOld] .
        have uniq:"unique_locations (Denvalue env)"
          using 2(1) typeSafeUnique by simp
        have epEnv:"ep $$ Contract env = Some (ct, dud)"
          using epOld cEq by simp
        show ?thesis
        proof(cases rule:lexpStorage[OF 2(1) subcase2(2) 2(3)])
          case 1
          then have inDenChanged:"(type.Storage (STValue t'), Storeloc locationChanged) |∈| fmran
(Denvalue env)" by simp
          show ?thesis
          proof(cases "i = locationChanged")
            case True
            have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage (STValue t'), Storeloc
locationChanged)"
              using uniqueLocs[OF uniq inDenI inDenChanged] True by simp
            then have tpEq:"tp = STValue t'" by simp
            show ?thesis using scont' True tpEq by (simp add: addrEq)
          next
            case False
            have ctChanged:"ct $$ locationChanged = Some (Var (STValue t'))"
              using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenChanged] .
            have n1:"¬TypedStoSubpref locationChanged i tp"
                and n2:"¬TypedStoSubpref i locationChanged (STValue t')"
              using methodVarsNoPref False epOld ctOld ctChanged by blast+
            have relChanged:"TypedStoSubpref locationChanged locationChanged (STValue t')
                             ∧ CompStoType (STValue t') (STValue t') locationChanged locationChanged"
              by simp
            show ?thesis
              using singleLocChanged_nonchanged_SCon[OF nonLocChanged False oldSCon n1 n2 relChanged]
              by (simp add: addrEq)
          qed
        next
          case sub2:(2 l t)
          have inDenL:"(type.Storage t, Storeloc l) |∈| fmran (Denvalue env)" using sub2 by simp
          have relL:"TypedStoSubpref locationChanged l t ∧ CompStoType t (STValue t') l
locationChanged" using sub2 by simp
          show ?thesis
```

```
        proof(cases "i = l")
          case True
          have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage t, Storeloc l)"
            using uniqueLocs[OF uniq inDenI inDenL] True by simp
          then have tpEq:"tp = t" by simp
          have comp:"CompStoType tp (STValue t') i locationChanged"
            using relL tpEq True by simp
          show ?thesis using SCon_value_write_imps_SCon[OF comp oldSCon scont' nonLocChanged]
            by (simp add: addrEq)
        next
          case False
          have ctL:"ct $$ l = Some (Var t)"
            using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenL] by simp
          have n1:"¬TypedStoSubpref l i tp"
              and n2:"¬TypedStoSubpref i l t"
            using methodVarsNoPref False epOld ctOld ctL by blast+
          have iNeChanged:"i ≠ locationChanged"
          proof
            assume "i = locationChanged"
            then have "TypedStoSubpref i l t" using relL by simp
            then show False using n2 by contradiction
          qed
          show ?thesis
            using singleLocChanged_nonchanged_SCon[OF nonLocChanged iNeChanged oldSCon n1 n2 relL]
            by (simp add: addrEq)
        qed
      next
        case (3 l' t l)
        have inDenStk:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue env)" using 3 by simp
        have ptrL:"accessStore l (Stack st) = Some (KStoptr l')" using 3 by simp
        have rel0:"TypedStoSubpref locationChanged l' t ∧ CompStoType t (STValue t') l'
locationChanged" using 3 by simp
        obtain tprnt lprnt where
            inDenPr:"(type.Storage tprnt, Storeloc lprnt) |∈| fmran (Denvalue env)"
            and compPr:"CompStoType tprnt t lprnt l'"
          using fiPtr_parent_from_fullyInitialised[OF 2(3) inDenStk ptrL] by blast
        have compPr2:"CompStoType tprnt (STValue t') lprnt locationChanged"
          using compPr rel0 CompStoType_trns by blast
        have relPr:"TypedStoSubpref locationChanged lprnt tprnt ∧ CompStoType tprnt (STValue t')
lprnt locationChanged"
          using compPr2 CompStoType_imps_TypedStoSubpref by blast
        show ?thesis
        proof(cases "i = lprnt")
          case True
          have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage tprnt, Storeloc lprnt)"
            using uniqueLocs[OF uniq inDenI inDenPr] True by simp
          then have tpEq:"tp = tprnt" by simp
          have comp:"CompStoType tp (STValue t') i locationChanged"
            using compPr2 tpEq True by simp
          show ?thesis using SCon_value_write_imps_SCon[OF comp oldSCon scont' nonLocChanged]
            by (simp add: addrEq)
        next
          case False
          have ctPr:"ct $$ lprnt = Some (Var tprnt)"
            using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenPr] .
          have n1:"¬TypedStoSubpref lprnt i tp"
              and n2:"¬TypedStoSubpref i lprnt tprnt"
            using methodVarsNoPref False epOld ctOld ctPr by blast+
          have iNeChanged:"i ≠ locationChanged"
          proof
            assume "i = locationChanged"
            then have "TypedStoSubpref i lprnt tprnt" using relPr by simp
            then show False using n2 by contradiction
          qed
```

```
            show ?thesis
              using singleLocChanged_nonchanged_SCon[OF nonLocChanged iNeChanged oldSCon n1 n2 relPr]
              by (simp add: addrEq)
          qed
        qed
      qed
    qed
  next
    have "Accounts st' = Accounts st" using subcase2(4) by simp
    then show "balanceTypes (Accounts st')" using 2(1) unfolding TypeSafe_def balanceTypes_def by
simp
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have "Stack st = Stack st'" using subcase2(4) by simp
    then show "lessThanTopLocs (Stack st')" using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by
simp
  next
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd" unfolding
typeCompat_def
    proof intros
      fix t l
      assume inDen:"(t, l) |∈| fmran (Denvalue env)"
      then show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t l"
      proof (cases l)
        case (Stackloc loc)
        then have "l ≠ Storeloc locationChanged" by simp
        then have same:"accessStore loc (Stack st) = accessStore loc (Stack st')" using subcase2(4) by
simp
        then show ?thesis
        proof(cases "accessStore loc (Stack st')")
          case None
          then show ?thesis using same 2(1) inDen Stackloc unfolding TypeSafe_def typeCompat_def by
fastforce
        next
          case (Some a)
          have samemem:"(Memory st) = Memory st'" using subcase2(4) by simp
          then show ?thesis
          proof(cases a)
            case (KValue x1)
            then show ?thesis using Some same 2(1) inDen Stackloc unfolding TypeSafe_def
typeCompat_def by fastforce
          next
            case (KCDptr x2)
            then show ?thesis using Some same 2(1) inDen Stackloc unfolding TypeSafe_def
typeCompat_def samemem by fastforce
          next
            case (KMemptr x3)
            then show ?thesis using Some same 2(1) inDen Stackloc unfolding TypeSafe_def
typeCompat_def samemem by fastforce
          next
            case (KStoptr x4)
            then have a20:"accessStore loc (Stack st) = Some (KStoptr x4)" using same Some by simp
            have a25:"denvalueCon (Stack st) (Memory st) (Storage st (Address env)) cd t l"
              using 2(1) unfolding TypeSafe_def typeCompat_def using inDen by simp
            then obtain struct where a30:"t = type.Storage struct" using a20 Stackloc by (simp
split:type.splits)
            then have a40:"SCon struct x4 (Storage st (Address env))" using a25 a20 Stackloc by (simp
split:type.splits)
            have tCont2v:"typeCon t2 v" using subcase2(1) extractValueType.simps 2(1)
              using exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st⦈)"
                  "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KValue v" "Value t2" g]
            using 2(3) unfolding fullyInitialised_def
```

```
            by (simp)
          have "v' = v" using convertSame[of t2 t' v v'] subcase2(3) by simp
          then have tCont'v:"typeCon t' v" using typeSafeConvert[of t2 v t'] subcase2(3) tCont2v by
simp
          have "SCon struct x4 (Storage st' (Address env))" using a40 a30 inDen
          proof(cases "TypedStoSubpref locationChanged x4 struct")
            case subloc:True
            have compPtr:"(∀ tp1 tp2 l1 l2 l1' l2' stl1 stl2.
                    (type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
                    (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
                    (l1 = Stackloc l1' ∧ accessStore l1' (Stack st) = Some (KStoptr stl1) ∨ l1 =
Storeloc stl1)
                    ∧ (l2 = Stackloc l2' ∧ accessStore l2' (Stack st) = Some (KStoptr stl2) ∨ l2 =
Storeloc stl2) ⟶
                    (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
              using 2(1) unfolding TypeSafe_def compPointers_def by auto
            then show ?thesis
            proof(cases rule:lexpStorage[OF 2(1) subcase2(2) 2(3)])
              case 11:1
              then have b40:"CompStoType struct (STValue t') x4 locationChanged"
                using inDen Stackloc Some KStoptr a20 a30 compPtr subloc by metis
              then show ?thesis
                using SCon_value_write_imps_SCon[OF _ a40 scont' nonLocChanged] by blast
            next
              case (2 l'' t'')
              have c3:"∀ l struct. ¬TypedStoSubpref locationChanged l struct ⟶ Storage st' (Address
env) $$ l = Storage st (Address env) $$ l"
                by (metis TypedStoSubpref_sameLoc nonLocChanged)
              have b40:"if TypedStoSubpref x4 l'' t'' then CompStoType t'' struct l'' x4
                            else if TypedStoSubpref l'' x4 struct then CompStoType struct t'' x4 l''
                              else True"
              using compPtr inDen Stackloc Some KStoptr a20 a30 2 by blast
              then show ?thesis using subloc 2
              proof(cases "x4 = l''")
                case True
                then have stt:"struct = t''" using b40
                  using CompStoType_sameLoc_sameType TypedStoSubpref_sameLoc by auto
                then show ?thesis
                proof(cases "l'' = locationChanged")
                  case t4:True
                  then show ?thesis using stt True scont'
                    using "2"(2) CompStoType_sameLoc_sameType by blast
                next
                  case False
                  have c0:"CompStoType struct (STValue t') x4 locationChanged" using 2(2) True stt
by simp
                  then have "∃y. hash l'' y = locationChanged" using subloc "2"(2) TypedStoSubpref_b
False by auto
                  then obtain i'' where c1:"(hash x4 i'' = locationChanged)" using
CompStoType_imps_subloc False True by blast
                  then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 by
blast
                qed
              next
                case x4Dif:False
                then show ?thesis
                proof(cases "TypedStoSubpref x4 l'' t''")
                  case d5:True
                  then have d7:"CompStoType t'' struct l'' x4" using b40 by simp
                  have d9:"CompStoType t'' (STValue t') l'' locationChanged" using 2(2) by simp
                  have d10:"CompStoType struct (STValue t') x4 locationChanged" using
CompStoType_subloc_type_transfer[of t'' t' l'' locationChanged struct x4] subloc d7 d9 by blast
                  then have d11:"∃x. hash l'' x = x4" using TypedStoSubpref_b x4Dif d5 by auto
```

```
                    then show ?thesis
                    proof(cases "l'' = locationChanged")
                      case True
                      then obtain i'' where d11:"hash locationChanged i'' = x4" using d11 by auto
                      then show ?thesis using subloc using TypedStoSubpref_hashes by auto
                    next
                      case False
                      then have "∃x. hash l'' x = locationChanged" using 2(2) TypedStoSubpref_b by
auto
                      then have "∃x. hash l'' x = x4" using d11 using hash_suffixes_associative by
auto
                      then show ?thesis
                      proof(cases "x4 = locationChanged")
                        case True
                        then have "struct = (STValue t')" using d10 by (simp add:
CompStoType_sameLoc_sameType)
                        then show ?thesis using True scont' by auto
                      next
                        case False
                        then obtain i'' where d11:"hash x4 i'' = locationChanged"
                          using subloc False TypedStoSubpref_b[of locationChanged x4 struct] by blast
                        then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11
d10 by blast
                      qed
                    qed
                  next
                    case f5:False
                    then show ?thesis
                    proof(cases "TypedStoSubpref l'' x4 struct")
                      case True
                      then have d5:"TypedStoSubpref l'' x4 struct" by simp
                      then have d7:"CompStoType struct t'' x4 l''" using b40 f5 by simp
                      then have d10:"∃x. hash x4 x = l''" using TypedStoSubpref_b x4Dif d5 by auto
                      then show ?thesis
                      proof(cases "l'' = locationChanged")
                        case True
                        then obtain i'' where d11:"hash x4 i'' = locationChanged" using d10 by auto
                        then have d15:"t'' = STValue t'" using 2(2) CompStoType_sameLoc_sameType True
by auto
                        then have c0:"CompStoType struct (STValue t') x4 locationChanged" using 2(2)
d15 d7 by fastforce
                        then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3
d11 by blast
                      next
                        case False
                        then have "∃x. hash l'' x = locationChanged" using 2(2) TypedStoSubpref_b by
auto
                        then have "∃x. hash x4 x = locationChanged" using d10 using
hash_suffixes_associative by auto
                        then obtain i'' where d11:"hash x4 i'' = locationChanged" by auto
                        then have "CompStoType struct (STValue t') x4 locationChanged" using
CompStoType_trns 2(2) d7 by blast
                        then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11
by blast
                      qed
                    next
                      case False
                      have "x4 ≠ locationChanged" using 2(2) f5 by blast
                      then show ?thesis using a40 False f5 2(2) singleLocChanged_nonchanged_SCon
                        using nonLocChanged by blast
                    qed
                  qed
                qed
              next
```

```
                case (3 l'' t'' l)
                have c3:"∀l struct. ¬TypedStoSubpref locationChanged l struct ⟶ Storage st' (Address
env) $$ l = Storage st (Address env) $$ l"
                  by (metis TypedStoSubpref_sameLoc nonLocChanged)
                then have b40:"if TypedStoSubpref x4 l'' t'' then CompStoType t'' struct l'' x4
                                else if TypedStoSubpref l'' x4 struct then CompStoType struct t'' x4 l''
                                  else True"
                  using compPtr inDen Stackloc Some KStoptr a20 a30 3 by blast
                then show ?thesis using subloc 2
                proof(cases "x4 = l''")
                  case True
                  then have stt:"struct = t''" using b40
                    using CompStoType_sameLoc_sameType TypedStoSubpref_sameLoc by auto
                  then show ?thesis
                  proof(cases "l'' = locationChanged")
                    case t4:True
                    then show ?thesis using stt True scont'
                      using "3"(3) CompStoType_sameLoc_sameType by blast
                  next
                    case False
                    have c0:"CompStoType struct (STValue t') x4 locationChanged" using 3(3) True stt
by simp
                    then have "∃y. hash l'' y = locationChanged" using subloc 3(3) TypedStoSubpref_b
False by auto
                    then obtain i'' where c1:"(hash x4 i'' = locationChanged)" using
CompStoType_imps_subloc False True by blast
                    then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 by
blast
                  qed
                next
                  case x4Dif:False
                  then show ?thesis
                  proof(cases "TypedStoSubpref x4 l'' t''")
                    case d5:True
                    then have d7:"CompStoType t'' struct l'' x4" using b40 by simp
                    have d9:"CompStoType t'' (STValue t') l'' locationChanged" using 3(3) by simp
                    have d10:"CompStoType struct (STValue t') x4 locationChanged" using
CompStoType_subloc_type_transfer[of t'' t' l'' locationChanged struct x4] subloc d7 d9 by blast
                    then have d11:"∃x. hash l'' x = x4" using TypedStoSubpref_b x4Dif d5 by auto
                    then show ?thesis
                    proof(cases "l'' = locationChanged")
                      case True
                      then obtain i'' where d11:"hash locationChanged i'' = x4" using d11 by auto
                      then show ?thesis using subloc using TypedStoSubpref_hashes by auto
                    next
                      case False
                      then have "∃x. hash l'' x = locationChanged" using 3(3) TypedStoSubpref_b by
auto
                      then have "∃x. hash l'' x = x4" using d11 using hash_suffixes_associative by
auto
                      then show ?thesis
                      proof(cases "x4 = locationChanged")
                        case True
                        then have "struct = (STValue t')" using d10 by (simp add:
CompStoType_sameLoc_sameType)
                        then show ?thesis using True scont' by auto
                      next
                        case False
                        then obtain i'' where d11:"hash x4 i'' = locationChanged"
                          using subloc False TypedStoSubpref_b[of locationChanged x4 struct] by blast
                        then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11
d10 by blast
                      qed
                    qed
```

```
                next
                  case f5:False
                  then show ?thesis
                  proof(cases "TypedStoSubpref l'' x4 struct")
                    case True
                    then have d5:"TypedStoSubpref l'' x4 struct" by simp
                    then have d7:"CompStoType struct t'' x4 l''" using b40 f5 by simp
                    then have d10:"∃x. hash x4 x = l''" using TypedStoSubpref_b x4Dif d5 by auto
                    then show ?thesis
                    proof(cases "l'' = locationChanged")
                      case True
                      then obtain i'' where d11:"hash x4 i'' = locationChanged" using d10 by auto
                      then have d15:"t'' = STValue t'" using 3(3) CompStoType_sameLoc_sameType True
by auto
                      then have c0:"CompStoType struct (STValue t') x4 locationChanged" using 3(3)
d15 d7 by fastforce
                      then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3
d11 by blast
                    next
                      case False
                      then have "∃x. hash l'' x = locationChanged" using 3(3) TypedStoSubpref_b by
auto
                      then have "∃x. hash x4 x = locationChanged" using d10 using
hash_suffixes_associative by auto
                      then obtain i'' where d11:"hash x4 i'' = locationChanged" by auto
                      then have "CompStoType struct (STValue t') x4 locationChanged" using
CompStoType_trns 3(3) d7 by blast
                      then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11
by blast
                    qed
                  next
                    case False
                    have "x4 ≠ locationChanged" using f5
                      using "3"(3) by auto
                    then show ?thesis using a40 False f5 3(3) singleLocChanged_nonchanged_SCon
                      using nonLocChanged by blast
                  qed
                qed
              qed
            qed
          next
            case f1:False
            then show ?thesis using a40
            proof(induction struct arbitrary:x4)
              case (STArray x1 struct)
              have b10:"∀i<x1. SCon struct (hash x4 (ShowL_nat i)) (Storage st (Address env))"
                using STArray(3) using SCon.simps(2)[of x1 struct x4 "(Storage st (Address env))"] by
auto
              have b20:"∀i<x1. ¬ TypedStoSubpref locationChanged (hash x4 (ShowL_nat i)) struct
                                ∧ locationChanged ≠ hash x4 (ShowL_nat i)"
                using STArray(2) TypedStoSubpref.simps(2)[of locationChanged x4 x1 struct]
                using TypedStoSubpref_sameLoc by auto
              then have b30:"∀i<x1. ¬ TypedStoSubpref locationChanged (hash x4 (ShowL_nat i)) struct
" by auto
              have " (∀i<x1. SCon struct (hash x4 (ShowL_nat i)) (Storage st' (Address env)))"
              proof(intros)
                fix i assume b40: "i<x1"
                then have "¬TypedStoSubpref locationChanged (hash x4 (ShowL_nat i)) struct" using b30
by auto
                then show "SCon struct (hash x4 (ShowL_nat i)) (Storage st' (Address env))" using
STArray(1)[of
                    "(hash x4 (ShowL_nat i))"] b30 b10 b40 by simp
              qed
              then show ?case using SCon.simps(2)[of x1 struct x4 "Storage st' (Address env)"] by
```

```
simp
              next
                case (STMap x1 struct)
                then have " (∀i. typeCon x1 i ⟶ SCon struct (hash x4 i) (Storage st (Address env)))"
using SCon.simps(3)[of x1 struct x4 "(Storage st (Address env))"] by simp
                moreover have "(∀i. ¬typeCon x1 i ∨ ¬TypedStoSubpref locationChanged (hash x4 i)
struct ∧ locationChanged ≠ hash x4 i)"
                  using STMap(2) TypedStoSubpref.simps(3)[of locationChanged x4 x1 struct]
                  using TypedStoSubpref_sameLoc by auto
                ultimately show ?case using STMap(1) by auto
              next
                case (STValue x)
                then have c5:"(x4 ≠ locationChanged ∨ STValue t' ≠ STValue x)" using
Subpref.simps(1)[of x x4 "STValue t'" locationChanged] by auto
                then show ?case
                proof(cases "x4 ≠ locationChanged")
                  case True
                  then have c10:"Storage st' (Address env) $$ x4 = Storage st (Address env) $$ x4"
using subcase2 nonLocChanged by auto
                  then have " SCon (STValue x) x4 (Storage st (Address env))" using STValue by blast
                  then show ?thesis using c10
                    by (simp add: accessStorage_def)
                next
                  case False
                  then have "x4 = locationChanged" by simp
                  then have "STValue t' = STValue x" using subcase2 Stackloc KStoptr inDen
lexpStorage[of env st cd lv g locationChanged t' g'] Some f1 2(1) a20 a30
                    by (metis STValue.prems(1) TypedStoSubpref.simps(1))
                  then show ?thesis using False c5 by auto
                qed
              qed
            qed
            then show ?thesis using Stackloc inDen KStoptr Some a30 subcase2 by simp
          qed
        qed
      next
        case (Storeloc x2)
        then have a20:"case t of type.Storage typ ⇒ SCon typ x2 (Storage st (Address env)) | _ ⇒
False"
          using 2(1) unfolding TypeSafe_def typeCompat_def using inDen by force
        then obtain struct where a30:"t = type.Storage struct" using a20 Storeloc by (simp
split:type.splits)
        then have a40:"SCon struct x2 (Storage st (Address env))" using a20 Storeloc by (simp
split:type.splits)
        have tCont2v:"typeCon t2 v" using subcase2(1) extractValueType.simps 2(1)
          using exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env
cd st⦈)" "(state.Gas
              st - costs (ASSIGN lv ex) env cd st)" "KValue v" "Value t2" g]
          using 2(3) unfolding fullyInitialised_def by (simp)
        have "v' = v" using convertSame[of t2 t' v v'] subcase2(3) by simp
        then have tCont'v:"typeCon t' v" using typeSafeConvert[of t2 v t'] subcase2(3) tCont2v by simp
        have "SCon struct x2 (Storage st' (Address env))" using a40 a30 inDen
        proof(cases "TypedStoSubpref locationChanged x2 struct")
          case subloc:True
          have compPtr:"(∀tp1 tp2 l1 l2 l1' l2' stl1 stl2.
                      (type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
                      (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
                      (l1 = Stackloc l1' ∧ accessStore l1' (Stack st) = Some (KStoptr stl1) ∨ l1 =
Storeloc stl1)
                      ∧ (l2 = Stackloc l2' ∧ accessStore l2' (Stack st) = Some (KStoptr stl2) ∨ l2 =
Storeloc stl2) ⟶
                      (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
            using 2(1) unfolding TypeSafe_def compPointers_def by auto
```

```
          then show ?thesis
          proof(cases rule:lexpStorage[OF 2(1) subcase2(2) 2(3)])
            case 1
            then have b40:"CompStoType struct (STValue t') x2 locationChanged"
              using inDen Storeloc a20 a30 1 compPtr subloc by metis
            then show ?thesis
              using SCon_value_write_imps_SCon[OF _ a40 scont' nonLocChanged] by blast
          next
            case (2 l'' t'')
            have c3:"∀l struct. ¬TypedStoSubpref locationChanged l struct ⟶ Storage st' (Address
env) $$ l = Storage st (Address env) $$ l"
              by (metis TypedStoSubpref_sameLoc nonLocChanged)
            then have b40:"if TypedStoSubpref x2 l'' t'' then CompStoType t'' struct l'' x2
                           else if TypedStoSubpref l'' x2 struct then CompStoType struct t'' x2 l''
                             else True"
              using compPtr inDen Storeloc a20 a30 2 by blast
            then show ?thesis using subloc 2
            proof(cases "x2 = l''")
              case True
              then have stt:"struct = t''" using b40
                using CompStoType_sameLoc_sameType TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l'' = locationChanged")
                case t4:True
                then show ?thesis using stt True scont'
                  using "2"(2) CompStoType_sameLoc_sameType by blast
              next
                case False
                have c0:"CompStoType struct (STValue t') x2 locationChanged" using 2(2) True stt by
simp
                then have "∃y. hash l'' y = locationChanged" using subloc "2"(2) TypedStoSubpref_b
False by auto
                then obtain i'' where c1:"(hash x2 i'' = locationChanged)" using
CompStoType_imps_subloc False True by blast
                then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 by blast
              qed
            next
              case x4Dif:False
              then show ?thesis
              proof(cases "TypedStoSubpref x2 l'' t''")
                case d5:True
                then have d7:"CompStoType t'' struct l'' x2" using b40 by simp
                have d9:"CompStoType t'' (STValue t') l'' locationChanged" using 2(2) by simp
                have d10:"CompStoType struct (STValue t') x2 locationChanged" using
CompStoType_subloc_type_transfer[of t'' t' l'' locationChanged struct x2] subloc d7 d9 by blast
                then have d11:"∃x. hash l'' x = x2" using TypedStoSubpref_b x4Dif d5 by auto
                then show ?thesis
                proof(cases "l'' = locationChanged")
                  case True
                  then obtain i'' where d11:"hash locationChanged i'' = x2" using d11 by auto
                  then show ?thesis using subloc using TypedStoSubpref_hashes by auto
                next
                  case False
                  then have "∃x. hash l'' x = locationChanged" using 2(2) TypedStoSubpref_b by auto
                  then have "∃x. hash l'' x = x2" using d11 using hash_suffixes_associative by auto
                  then show ?thesis
                  proof(cases "x2 = locationChanged")
                    case True
                    then have "struct = (STValue t')" using d10 by (simp add:
CompStoType_sameLoc_sameType)
                    then show ?thesis using True scont' by auto
                  next
                    case False
                    then obtain i'' where d11:"hash x2 i'' = locationChanged"
```

```
                    using subloc False TypedStoSubpref_b[of locationChanged x2 struct] by blast
                  then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11 d10
by blast
                qed
              qed
            next
              case f5:False
              then show ?thesis
              proof(cases "TypedStoSubpref l'' x2 struct")
                case True
                then have d5:"TypedStoSubpref l'' x2 struct" by simp
                then have d7:"CompStoType struct t'' x2 l''" using b40 f5 by simp
                then have d10:"∃x. hash x2 x = l''" using TypedStoSubpref_b x4Dif d5 by auto
                then show ?thesis
                proof(cases "l'' = locationChanged")
                  case True
                  then obtain i'' where d11:"hash x2 i'' = locationChanged" using d10 by auto
                  then have d15:"t'' = STValue t'" using 2(2) CompStoType_sameLoc_sameType True by
auto
                  then have c0:"CompStoType struct (STValue t') x2 locationChanged" using 2(2) d15
d7 by fastforce
                  then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 d11
by blast
                next
                  case False
                  then have "∃x. hash l'' x = locationChanged" using 2(2) TypedStoSubpref_b by auto
                  then have "∃x. hash x2 x = locationChanged" using d10 using
hash_suffixes_associative by auto
                  then obtain i'' where d11:"hash x2 i'' = locationChanged" by auto
                  then have "CompStoType struct (STValue t') x2 locationChanged" using
CompStoType_trns 2(2) d7 by blast
                  then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11 by
blast
                qed
              next
                case False
                have "x2 ≠ locationChanged" using 2(2) f5 by blast
                then show ?thesis using a40 False f5 2(2) singleLocChanged_nonchanged_SCon
                  using nonLocChanged by blast
              qed

            qed
          qed
        next
          case (3 l'' t'' l)
          have c3:"∀l struct. ¬TypedStoSubpref locationChanged l struct ⟶ Storage st' (Address
env) $$ l = Storage st (Address env) $$ l"
            by (metis TypedStoSubpref_sameLoc nonLocChanged)
          then have b40:"if TypedStoSubpref x2 l'' t'' then CompStoType t'' struct l'' x2
                           else if TypedStoSubpref l'' x2 struct then CompStoType struct t'' x2 l''
                             else True"
            using compPtr inDen Storeloc a20 a30 3 by blast
          then show ?thesis using subloc 2
          proof(cases "x2 = l''")
            case True
            then have stt:"struct = t''" using b40
              using CompStoType_sameLoc_sameType TypedStoSubpref_sameLoc by auto
            then show ?thesis
            proof(cases "l'' = locationChanged")
              case t4:True
              then show ?thesis using stt True scont'
                using "3"(3) CompStoType_sameLoc_sameType by blast
            next
              case False
```

```
              have c0:"CompStoType struct (STValue t') x2 locationChanged" using 3(3) True stt by
simp
              then have "∃y. hash l'' y = locationChanged" using subloc 3(3) TypedStoSubpref_b False
by auto
              then obtain i'' where c1:"(hash x2 i'' = locationChanged)" using
CompStoType_imps_subloc False True by blast
              then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 by blast
            qed
          next
            case x4Dif:False
            then show ?thesis
            proof(cases "TypedStoSubpref x2 l'' t''")
              case d5:True
              then have d7:"CompStoType t'' struct l'' x2" using b40 by simp
              have d9:"CompStoType t'' (STValue t') l'' locationChanged" using 3(3) by simp
              have d10:"CompStoType struct (STValue t') x2 locationChanged" using
CompStoType_subloc_type_transfer[of t'' t' l'' locationChanged struct x2] subloc d7 d9 by blast
              then have d11:"∃x. hash l'' x = x2" using TypedStoSubpref_b x4Dif d5 by auto
              then show ?thesis
              proof(cases "l'' = locationChanged")
                case True
                then obtain i'' where d11:"hash locationChanged i'' = x2" using d11 by auto
                then show ?thesis using subloc using TypedStoSubpref_hashes by auto
              next
                case False
                then have "∃x. hash l'' x = locationChanged" using 3(3) TypedStoSubpref_b by auto
                then have "∃x. hash l'' x = x2" using d11 using hash_suffixes_associative by auto
                then show ?thesis
                proof(cases "x2 = locationChanged")
                  case True
                  then have "struct = (STValue t')" using d10 by (simp add:
CompStoType_sameLoc_sameType)
                  then show ?thesis using True scont' by auto
                next
                  case False
                  then obtain i'' where d11:"hash x2 i'' = locationChanged"
                    using subloc False TypedStoSubpref_b[of locationChanged x2 struct] by blast
                  then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11 d10
by blast
                qed
              qed
            next
              case f5:False
              then show ?thesis
              proof(cases "TypedStoSubpref l'' x2 struct")
                case True
                then have d5:"TypedStoSubpref l'' x2 struct" by simp
                then have d7:"CompStoType struct t'' x2 l''" using b40 f5 by simp
                then have d10:"∃x. hash x2 x = l''" using TypedStoSubpref_b x4Dif d5 by auto
                then show ?thesis
                proof(cases "l'' = locationChanged")
                  case True
                  then obtain i'' where d11:"hash x2 i'' = locationChanged" using d10 by auto
                  then have d15:"t'' = STValue t'" using 3(3) CompStoType_sameLoc_sameType True by
auto
                  then have c0:"CompStoType struct (STValue t') x2 locationChanged" using 3(3) d15
d7 by fastforce
                  then show ?thesis using stvalueLocationsInduct a40 scont' c0 nonLocChanged c3 d11
by blast
                next
                  case False
                  then have "∃x. hash l'' x = locationChanged" using 3(3) TypedStoSubpref_b by auto
                  then have "∃x. hash x2 x = locationChanged" using d10 using
hash_suffixes_associative by auto
```

```
                then obtain i'' where d11:"hash x2 i'' = locationChanged" by auto
                then have "CompStoType struct (STValue t') x2 locationChanged" using
CompStoType_trns 3(3) d7 by blast
                then show ?thesis using stvalueLocationsInduct a40 scont' nonLocChanged c3 d11 by
blast
              qed
            next
              case False
              have "x2 ≠ locationChanged" using 3(3) f5 by blast
              then show ?thesis using a40 False f5 3(3) singleLocChanged_nonchanged_SCon
                using nonLocChanged by blast
            qed
          qed
        qed
      qed
    next
      case f1:False
      then show ?thesis using a40
      proof(induction struct arbitrary:x2)
        case (STArray x1 struct)
        have b10:"∀i<x1. SCon struct (hash x2 (ShowL_nat i)) (Storage st (Address env))"
          using STArray(3) using SCon.simps(2)[of x1 struct x2 "(Storage st (Address env))"] by
auto
        have b20:"∀i<x1. ¬ TypedStoSubpref locationChanged (hash x2 (ShowL_nat i)) struct
                                ∧ locationChanged ≠ hash x2 (ShowL_nat i)"
          using STArray(2) TypedStoSubpref.simps(2)[of locationChanged x2 x1 struct]
          using TypedStoSubpref_sameLoc by auto
        then have b30:"∀i<x1. ¬ TypedStoSubpref locationChanged (hash x2 (ShowL_nat i)) struct "
by auto
        have " (∀i<x1. SCon struct (hash x2 (ShowL_nat i)) (Storage st' (Address env)))"
        proof(intros)
          fix i assume b40: "i<x1"
          then have "¬TypedStoSubpref locationChanged (hash x2 (ShowL_nat i)) struct" using b30 by
auto
          then show "SCon struct (hash x2 (ShowL_nat i)) (Storage st' (Address env))" using
STArray(1)[of "(hash x2 (ShowL_nat i))"] b30 b10 b40 by simp
        qed
        then show ?case using SCon.simps(2)[of x1 struct x2 "Storage st' (Address env)"] by simp
      next
        case (STMap x1 struct)
        then have " (∀i. typeCon x1 i ⟶ SCon struct (hash x2 i) (Storage st (Address env)))"
using SCon.simps(3)[of x1 struct x2 "(Storage st (Address env))"] by simp
        moreover have "(∀i. ¬typeCon x1 i ∨ ¬TypedStoSubpref locationChanged (hash x2 i) struct
∧ locationChanged ≠ hash x2 i)"
          using STMap(2) TypedStoSubpref.simps(3)[of locationChanged x2 x1 struct]
          using TypedStoSubpref_sameLoc by auto
        ultimately show ?case using STMap(1) by auto
      next
        case (STValue x)
        then have c5:"(x2 ≠ locationChanged ∨ STValue t' ≠ STValue x)" using Subpref.simps(1)[of
x x2 "STValue t'" locationChanged] by auto
        then show ?case
        proof(cases "x2 ≠ locationChanged")
          case True
          then have c10:"Storage st' (Address env) $$ x2 = Storage st (Address env) $$ x2" using
subcase2 nonLocChanged by auto
          then have " SCon (STValue x) x2 (Storage st (Address env))" using STValue by blast
          then show ?thesis using c10
            by (simp add: accessStorage_def)
        next
          case False
          then have "x2 = locationChanged" by simp
          then have "STValue t' = STValue x" using subcase2 Storeloc inDen lexpStorage[of env st
cd lv g locationChanged t' g'] f1 2(1) a20 a30
```

```
              by (metis STValue.prems(1) TypedStoSubpref.simps(1))
            then show ?thesis using False c5 by auto
          qed
        qed
      qed
      then show ?thesis
        using Storeloc a30 by auto
    qed
  qed
  then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd" using
subcase2 by auto
 next
  show "lessThanTopLocs cd" using 2 unfolding TypeSafe_def by simp
 next
  have "Memory st = Memory st'" using subcase2(4) by simp
  then show "lessThanTopLocs (Memory st')" using 2 unfolding TypeSafe_def by simp
 next
  have "envAddressesWellFormed env" using 2 unfolding TypeSafe_def by simp
  then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+

 next
  show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using subcase2 by simp
 next
  have "Accounts st'= Accounts st" using subcase2 by auto
  then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) subcase2(4) unfolding
fullyInitialised_def updateStore_def accessStore_def by auto
 next
  have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
    using subcase2(4) unfolding updateStore_def accessStore_def by auto
  show "denvalueTypeCorrectness env (Stack st') (Memory st')"
    unfolding denvalueTypeCorrectness_def
  proof intros
    fix t l ptr_loc
    assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
   accessStore l (Stack st') = Some (KMemptr ptr_loc)"
    then have "(case t of
     MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
      using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
    moreover have "Memory st = Memory st'" using subcase2(4) by simp
    ultimately show "case t of
   MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
   | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
      by metis
  qed
 next
  show "subPrefixStructuralConsistency (Memory st')"
    using 2(1) subcase2(4) unfolding TypeSafe_def by simp
 next
  show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using subcase2(4) by simp
 next
  show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
    using subcase2 by simp
 next
  show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
    using subcase2 by simp
 next
  show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
    using subcase2 by simp
 next
```

```
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using subcase2 by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using subcase2 by simp
  qed
qed
```

Assignment case 3: a value assigned to a memory location.

```
lemma TypeSafe_Assign_3:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv ex) env cd st) = Normal ((KValue v, Value t), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LMemloc l, type.Memory (MTValue t')),g')"
    and "convert t t' v = Some v'"
    and "st' = st(|state.Gas := g', Memory := updateStore l (MValue v') (Memory st)|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 3 = assms(4-7)
  have sameStack:"Stack st' = Stack st" using 3(4) by auto
  have sameStorage:"Storage st' = Storage st " using 3 by auto
  have sameAccounts:"Accounts st = Accounts st'" using 3 by simp
  have temp:"TypeSafe env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|))) (Memory (st(|Gas := g|)))
(Storage (st(|Gas := g|))) cd"
    using 2(1) by simp
  have ttt:"fullyInitialised env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|)))" using 2(3) unfold-
ing fullyInitialised_def by simp
  show ?thesis
  proof(cases rule:lexpIndexMem[OF 3(2) temp ttt])
    case lInfo:(1 x21 x22 tp tParent l' l'' prnt len' arr' i)

    have nonLocChanged:"∀locs. locs ≠ l ⟶ accessStore locs (Memory st) = accessStore locs (Memory
st')"
      using 3 unfolding updateStore_def accessStore_def by simp

    have TCsrc:"typeCon t (extractValueType (KValue v)) ∧ (∃xx. KValue v = KValue xx)"
      using 2(1) 3(1) exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st|))"
          "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KValue v" "Value t" g]
      using 2(3) unfolding fullyInitialised_def
      by (auto split:type.splits if_splits )


    obtain len subT where tParentType:"tParent = MTArray len subT" using lInfo by blast
    then have compType:"CompMemType (Memory (st(|Gas := g|))) len subT (MTValue t') l'' l"
        and lsublocs:"l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = (MTValue t') ∧ MCon (MTArray len'
arr') (Memory (st)) prnt"
        and lsublocs3:"(prnt = l'' ∧ len = len' ∧ arr' = subT ∨ CompMemType (Memory (st)) len subT
(MTArray len' arr') l'' prnt)"
      using lInfo 3(3) by fastforce+

    then have lsublocs2:" CompMemType (Memory (st)) len' arr' (MTValue t') prnt l"
      using 3(3) CompMemType.simps(2) by auto
    then have bb9:"∀subT subloc. CompMemType (Memory (st)) len' arr' subT prnt subloc ∧ subloc = l
                              ⟶ subT = (MTValue t')"
      using CompMemTypeSameLocsSameType lsublocs by blast

    have mconPrnt:"MCon (MTArray len' (MTValue t')) (Memory st) prnt" using lsublocs by auto
    have ldef:"l = hash prnt (ShowL_nat i) ∧ i < len'" using lsublocs by auto
```

```
    have t6:"(type.Memory tParent, Stackloc l') |∈| fmran (Denvalue env)" using lInfo by blast
    have t7:" MCon (MTArray len subT) (Memory (st)) l''" using lInfo
      using tParentType by auto
    then have mconlOld:"MCon (MTValue t') (Memory st) l" using 3(3) lInfo
      using CompTypeRemainsMCon lsublocs lsublocs2 by blast
    have "MCon (MTArray len' arr') (Memory st) prnt" using lsublocs3
      using lsublocs by blast
    then have mcPrntNew:"MCon (MTArray len' arr') (Memory st') prnt"
    proof -
      have lenNotZero:"len' ≠ 0" using lsublocs by auto
      have prntNotL:"prnt ≠ l" using ldef
        by (metis hash_inequality)
      then have p2:"(∃p. accessStore prnt (Memory st') = Some (MPointer p)) ∨ accessStore prnt (Memory
st') = None"
        using mconPrnt lenNotZero nonLocChanged by simp
      have "∀i<len'.
          (case accessStore (hash prnt (ShowL_nat i)) (Memory st') of None ⇒ False
           | Some (MValue val) ⇒ (case arr' of MTArray l a ⇒ False | MTValue typ ⇒ MCon arr' (Memory
st') (hash prnt (ShowL_nat i)))
           | Some (MPointer loc2) ⇒ (case arr' of MTArray len' arr'a ⇒ MCon arr' (Memory st') loc2 |
MTValue Types ⇒ False))
         "
      proof(intros)
        fix i1 assume in1:"i1<len'"
        show "case accessStore (hash prnt (ShowL_nat i1)) (Memory st') of None ⇒ False
        | Some (MValue val) ⇒ (case arr' of MTArray l a ⇒ False | MTValue typ ⇒ MCon arr' (Memory
st') (hash prnt (ShowL_nat i1)))
        | Some (MPointer loc2) ⇒ (case arr' of MTArray len' arr'a ⇒ MCon arr' (Memory st') loc2 |
MTValue Types ⇒ False)"
        proof(cases "i1 = i")
          case True
          then have "(hash prnt (ShowL_nat i1)) = l" using ldef by simp
          then have ac:"accessStore (hash prnt (ShowL_nat i1)) (Memory st') = Some((MValue v'))" using
3 by simp
          have "MCon (MTValue t') (Memory st) (hash prnt (ShowL_nat i1))"
            using ‹hash prnt (ShowL_nat i1) = l› lsublocs mconlOld by auto
          then have "typeCon t' v'" unfolding MCon.simps
            by (metis "3"(3) TCsrc convertSame extractValueType.simps(1) typeSafeConvert)
          then show ?thesis
            by (simp add: ‹accessStore (hash prnt (ShowL_nat i1)) (Memory st') = Some (MValue v')›
lsublocs)
        next
          case False
          then have sameAC:"accessStore (hash prnt (ShowL_nat i1)) (Memory st') = accessStore (hash prnt
              (ShowL_nat i1)) (Memory st)" using ldef nonLocChanged in1
            by (metis Read_Show_nat'_id hash_never_equal_suffix)
          then obtain ptr where ptrDef:"accessStore (hash prnt (ShowL_nat i1)) (Memory st) = Some
(MValue ptr)" using in1
            using lsublocs MCon_sub_MTVal_imps_val by blast
          then have ptrMC:"case accessStore (hash prnt (ShowL_nat i)) (Memory st) of None ⇒ False |
Some
              (MValue xa) ⇒ typeCon t' xa | Some (MPointer t) ⇒ False"
            using in1 mconPrnt unfolding MCon.simps
            using lsublocs mconlOld MCon.simps(1) by blast
          then show ?thesis using sameAC
            using in1 lsublocs ptrDef by force
        qed
      qed
      then show ?thesis using lenNotZero p2 by simp
    qed

    have "(type.Memory (MTArray len subT), Stackloc l') |∈| fmran (Denvalue env) ∧ accessStore l'
(Stack st) = Some (KMemptr l'')"
      using lInfo(2,3,4) tParentType by simp
```

```
    have cmpO_imps_new:"∀x11 x12 stl2 dt prnt. CompMemType (Memory st) x11 x12 dt stl2 prnt
          ⟶ CompMemType (Memory st') x11 x12 dt stl2 prnt"
    proof intros
      fix x11 x12 stl2 dt prnt
      assume "CompMemType (Memory st) x11 x12 dt stl2 prnt"
      then show "CompMemType (Memory st') x11 x12 dt stl2 prnt"
      proof(induction x12 arbitrary:x11 stl2)
        case (MTArray x1 x12)
        then obtain ii ll where iiDef: "ii<x11 ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st) =
Some (MPointer ll)
                ∧ (ll = prnt ∧ MTArray x1 x12 = dt ∨ CompMemType (Memory st) x1 x12 dt ll prnt)"
          unfolding CompMemType.simps by blast
        then show ?case
        proof(cases "ll = prnt")
          case True
          then show ?thesis using MTArray.prems iiDef
            by (metis MCon_sub_MTVal_imps_val MTArray.IH memoryvalue.distinct(1) lsublocs nonLocChanged
option.inject CompMemType.simps(2))
        next
          case False
          then show ?thesis
            by (metis MCon_sub_MTVal_imps_val MTArray.IH memoryvalue.distinct(1) iiDef lsublocs
nonLocChanged option.inject CompMemType.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case unfolding CompMemType.simps by simp
      qed

    qed

    have nToO_neg:"∀x11 x12 stl2 prnt . ¬TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt ⟶
¬TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
    proof intros
      fix x11 x12 stl2 prnt
      assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt"
      show "¬ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
      proof
        assume in2:"TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
        then show False using in1
        proof(induction x12 arbitrary:x11 stl2)
          case (MTArray x1 x12)
          then obtain ii ll where iiDef:"ii<x11 ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st) =
Some
              (MPointer ll) ∧ (ll = prnt ∨ TypedMemSubPrefPtrs (Memory st) x1 x12 ll prnt)"
            unfolding TypedMemSubPrefPtrs.simps by blast
          then show ?case
          proof(cases "ll = prnt")
            case True
            then have "(hash stl2 (ShowL_nat ii)) ≠ l" using iiDef mconlOld by auto
            then show ?thesis using True iiDef nonLocChanged
              using MTArray.prems(2) by auto
          next
            case False
            then show ?thesis using MTArray iiDef
              by (metis MCon_sub_MTVal_imps_val memoryvalue.distinct(1) lsublocs nonLocChanged
option.inject TypedMemSubPrefPtrs.simps(2))
          qed
        next
          case (MTValue x)
          then show ?case by simp
        qed
```

```
      qed
    qed


    have nToO:"∀ len2 arr2 stl2 stl1. TypedMemSubPrefPtrs (Memory st') len2 arr2 stl2 stl1 ⟶
TypedMemSubPrefPtrs (Memory st) len2 arr2 stl2 stl1"
    proof intros
      fix len2 arr2 stl2 stl1
      assume "TypedMemSubPrefPtrs (Memory st') len2 arr2 stl2 stl1"
      then show "TypedMemSubPrefPtrs (Memory st) len2 arr2 stl2 stl1 "
      proof(induction arr2 arbitrary:len2 stl2)
        case (MTArray x1 arr2)
        then obtain ii ll where iidef:"ii<len2 ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st') =
Some (MPointer ll) ∧
                                  (ll = stl1 ∨ TypedMemSubPrefPtrs (Memory st') x1 arr2 ll stl1)"
          unfolding TypedMemSubPrefPtrs.simps by auto
        then show ?case
        proof(cases "ll = stl1")
          case True
          then show ?thesis
            by (metis MCon_sub_MTVal_imps_val memoryvalue.distinct(1) iidef lsublocs mcPrntNew
nonLocChanged option.inject TypedMemSubPrefPtrs.simps(2))
        next
          case False
          then show ?thesis
            by (metis MCon_sub_MTVal_imps_val MTArray.IH memoryvalue.distinct(1) iidef lsublocs
mcPrntNew nonLocChanged option.inject TypedMemSubPrefPtrs.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed


    have mcl:"MCon (MTValue t') (Memory st') l" using mconlOld 3(4)
      using CompTypeRemainsMCon cmpO_imps_new lsublocs2 mcPrntNew by blast

    show ?thesis unfolding TypeSafe_def StateInvariant_def
    proof intros
      show "AddressTypes (Accounts st')" using 2(1) 3 unfolding TypeSafe_def by simp
    next
      show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
      then show "compPointers (Stack st') (Denvalue env)" using sameStack sameStorage by simp
    next
      show "safeContract (Accounts st') (Storage st')" using sameStorage sameAccounts using 2(1) un-
folding TypeSafe_def safeContract_def by auto
    next
      show "balanceTypes (Accounts st')" using 3 using 2(1) unfolding TypeSafe_def by simp
    next
      have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
      then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
    next
      show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have *:"((∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore
loc (Stack st) = None) ∧
            (∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
        using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
      have **:"Toploc (Stack st) = Toploc (Stack st')" using 3 unfolding updateStore_def by auto
      show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
```

```
      proof intros

        fix tloc loc
        assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
        then show "accessStore loc (Stack st') = None" using *
          by (simp add: sameStack)
      next
        fix loc y
        assume h1:" accessStore loc (Stack st') = Some y"
        then show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
          by (metis sameStack)
      qed
    next
      show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
    next
      have a12:"Toploc (Memory st) = Toploc (Memory st')" using 3 unfolding updateStore_def by auto
      have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp
      have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
      show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have b20:"Toploc (Memory st) ≤ tloc" using a12 tloc a12
          by force

        then show "accessStore loc (Memory st') = None "
          by (metis MCon_imps_Some a15 b10 lessThanSome_imps_Locs lessThanSome_imps_Locs_hash
lessThanTopLocs_def linorder_not_less mconlOld nonLocChanged)
      next
        fix loc y
        assume "accessStore loc (Memory st') = Some y "
        then show "∃tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
          by (metis MConAccessSame.simps(1) SameMCon_imps_MConAccessSame a12 a15 lessThanTopLocs_def
mconlOld nonLocChanged)
      qed
    next
      show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
        unfolding typeCompat_def
      proof intros
        fix tLook lLook
        assume inDen:" (tLook, lLook) |∈| fmran (Denvalue env)"
        show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd tLook lLook"
        proof(cases lLook)
          case (Stackloc x1)
          then show ?thesis
          proof(cases "accessStore x1 (Stack st')")
            case None
            then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc by force
          next
            case (Some a)
            then show ?thesis
            proof(cases a)
              case (KValue x1')
              then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by force
            next
              case (KCDptr x2)
              then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by force
            next
              case (KMemptr x3)
```

```
            then obtain struct where stT: "tLook = type.Memory struct" using 2(1) unfolding
TypeSafe_def typeCompat_def using sameStorage inDen sameStack Stackloc Some
              by (cases tLook; fastforce)
            have structss:"(case struct of
    MTArray len arr ⇒
      (∀ i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
    | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              using 2(1) inDen KMemptr Some Stackloc stT 3(4)
              unfolding TypeSafe_def denvalueTypeCorrectness_def by simp
            then have structss:"(case struct of
    MTArray len arr ⇒
      (∀ i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
    | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              by (metis (no_types, lifting) )
            have mcold:"MCon struct (Memory st) x3" using stT
              by (metis 2(1) KMemptr Some Stackloc inDen sameMemTSafe sameStack)
            then have "MCon struct (Memory st') x3" using structss
            proof(induction struct arbitrary:x3)
              case (MTArray x11 x12)
              have oldexp:"(∀ i<x11.
                    case accessStore (hash x3 (ShowL_nat i)) (Memory st) of None ⇒ False
                    | Some (MValue val) ⇒
                        (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 (Memory st) (hash
x3 (ShowL_nat i)))
                    | Some (MPointer loc2) ⇒
                        (case x12 of MTArray len' arr' ⇒ MCon x12 (Memory st) loc2 | MTValue val ⇒
False)) ∧
                (∃ p. accessStore x3 (Memory st) = Some (MPointer p) ∨ accessStore x3 (Memory st) =
None)"
                using MTArray.prems(1) unfolding MCon.simps by simp
              then have notL:"x3 ≠ l" using lInfo(7) by auto
              then have conc1:"(∃ p. accessStore x3 (Memory st') = Some (MPointer p) ∨ accessStore x3
(Memory st') = None)"
                using 3(4) oldexp unfolding accessStore_def updateStore_def by simp

              then have subs:"(∀ i<x11. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some
x12)"
                using structss MTArray by simp

              have "(∀ i<x11.
                    case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
                    | Some (MValue val) ⇒
                        (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 (Memory st')
(hash x3 (ShowL_nat i)))
                    | Some (MPointer loc2) ⇒
                        (case x12 of MTArray len' arr' ⇒ MCon x12 (Memory st') loc2 | MTValue val ⇒
False))"
              proof(intros)
                fix i assume idef:"i<x11"
                then consider (val1) val typw where "accessStore (hash x3 (ShowL_nat i)) (Memory st)
= Some (MValue val)
                                        ∧ x12 = MTValue typw
                                        ∧ MCon x12 (Memory st) (hash x3 (ShowL_nat i))"
                  | (ptr1) ptr l1 a1 where "accessStore (hash x3 (ShowL_nat i)) (Memory st) = Some
(MPointer ptr)
                                       ∧ x12 = MTArray l1 a1
                                       ∧ MCon x12 (Memory st) ptr"
                  using oldexp
                  by (metis MCon_imps_sub_Mcon MTArray.prems(1) mtypes.distinct(1) gr_implies_not0
mcon_accessStore nat_neq_iff)
              then show "case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
      | Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon x12 (Memory st')
(hash x3 (ShowL_nat i)))
      | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12 (Memory st') loc2 | MTValue
```

```
val ⇒ False)"
                  proof(cases)
                    case val1
                    then show ?thesis
                    proof(cases "(hash x3 (ShowL_nat i)) = l")
                      case True
                      then have "x12 = (MTValue t')" using subs lInfo(6) idef by auto
                      then show ?thesis using 3(4) True
                        unfolding updateStore_def accessStore_def using idef mcl by simp
                    next
                      case False
                      then have sameACC:"accessStore (hash x3 (ShowL_nat i)) (Memory st) = accessStore
(hash x3 (ShowL_nat i)) (Memory st')"
                        using 3(4)unfolding updateStore_def accessStore_def by simp
                      then have "MCon x12 (Memory st') (hash x3 (ShowL_nat i))"
                        using val1 unfolding MCon.simps by auto
                      then show ?thesis using val1 sameACC by simp
                    qed
                  next
                    case ptr1
                    then have notl:"(hash x3 (ShowL_nat i)) ≠ l" using mconlOld unfolding MCon.simps
by auto

                    have " case x12 of MTArray len arr ⇒
    (∀ i<len. accessTypeStore (hash ptr (ShowL_nat i)) (Memory st) = Some arr)
  | MTValue val ⇒ accessTypeStore ptr (Memory st) = Some (MTValue val)"
                      using 2(1) unfolding TypeSafe_def subPrefixStructuralConsistency_def using
MTArray.prems
                      using ptr1 idef by fastforce
                    then have "MCon x12 (Memory st') ptr"
                      using MTArray.IH[of ptr] ptr1 by blast
                    then show ?thesis using ptr1 notl 3(4) unfolding accessStore_def updateStore_def
by simp
                  qed
                qed
                then show ?case using conc1 MTArray.prems(1) unfolding MCon.simps by simp
              next
                case (MTValue x2)
                then show ?case
                proof(cases "x3 = l")
                  case True
                  then have "x2 = t'" using lInfo(6) structss MTValue by simp
                  then show ?thesis using lInfo MTValue True
                    using CompTypeRemainsMCon cmpO_imps_new lsublocs2 mcPrntNew by blast
                next
                  case False
                  then have "accessStore x3 (Memory st) = accessStore x3 (Memory st')"
                    using 3(4) unfolding updateStore_def accessStore_def by simp
                  then show ?thesis using mcold MTValue unfolding MCon.simps by simp
                qed
              qed

              then show ?thesis using sameStorage inDen sameStack Stackloc Some KMemptr stT by simp
            next
              case (KStoptr x4)
              then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by (cases tLook; force)
            qed
          qed
        next
          case (Storeloc x2)
          then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage inDen
            by (metis denvalue.simps(6))
        qed
```

```
      qed
      then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
using sameStack by auto
    next
      have "Accounts st'= Accounts st" using 3 by auto
      then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 3 unfolding
fullyInitialised_def by auto
    next

      have sameStack:"Stack st = Stack st'" using 3 by simp
      have oldL:"∃ v''. accessStore l (Memory (st(|Gas := g|))) = Some (MValue v'')" using lInfo
        using MCon_sub_MTVal_imps_val by blast
    next
      have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l
(Stack st) = Some (KMemptr ptr_loc)"
        using 3(4) unfolding updateStore_def accessStore_def by auto
      have cc1:"∀ locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using 3(4) unfolding updateStore_def accessTypeStore_def by simp
      show "denvalueTypeCorrectness env (Stack st') (Memory st')"
        unfolding denvalueTypeCorrectness_def
      proof intros
        fix t l ptr_loc sub_loc
        assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
        then have old:"(
      (case t of MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) =
Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val)))"
          using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast

        then show "case t of MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i))
(Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"

        proof(cases t)
          case (MTArray x11 x12)
          then have o2:" ∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12"
            using old by simp
          then have conc1:"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some
x12)" using cc1 by simp
          then show ?thesis using MTArray cc1 old 3(4) by auto
        next
          case (MTValue x2)
          then show ?thesis using cc1 old by simp
        qed
      qed
    next
      have cc1:"∀ locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using 3(4) unfolding updateStore_def accessTypeStore_def by simp
      have old:"(∀ locs. (∃ t. accessStore locs (Memory st) = Some t) = (∃ tt. accessTypeStore locs
(Memory st) = Some tt))"
        using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
      show "SomeValSomeTyp (Memory st')" unfolding SomeValSomeTyp_def
      proof intros
        fix locs
        show "(∃ t. accessStore locs (Memory st') = Some t) = (∃ tt. accessTypeStore locs (Memory st') =
Some tt) "
        proof(cases "locs = l")
          case True
          then have "accessStore locs (Memory st') = Some (MValue v')"
            using 3 unfolding updateStore_def accessStore_def by simp
          moreover have "∃ v. accessStore locs (Memory st) = Some v "
            using mconlOld True old by auto
          ultimately show ?thesis using old cc1 by simp
```

```
        next
          case False
          then have "accessStore locs (Memory st) = accessStore locs (Memory st')"
            using 3 unfolding updateStore_def accessStore_def by simp
          then show ?thesis using cc1 old by metis
        qed
      qed

    next
      have old:"subPrefixStructuralConsistency (Memory st)"
        using 2(1) unfolding TypeSafe_def by blast
      show "subPrefixStructuralConsistency (Memory st')"
        unfolding subPrefixStructuralConsistency_def
      proof intros
        fix locs tp
        assume in1:"accessTypeStore locs (Memory st') = Some tp"
        have allTyp:"∀ locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
          using 3 unfolding updateStore_def accessTypeStore_def by auto
        have notNo:"accessStore locs (Memory st) ≠ None"
        proof -
          have "accessTypeStore locs (Memory st) = Some tp"
            using in1 allTyp by simp
          then have "∃ v. accessStore locs (Memory st) = Some v"
            using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by simp
          then show ?thesis by simp
        qed
        then have old':"(case accessStore locs (Memory st) of Some (MValue v) ⇒
          ∃ val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs (Memory st) = Some
tp
      | Some (MPointer p) ⇒
          ∃ len arr.
             MCon tp (Memory st) p ∧
             tp = MTArray len arr ∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some
arr))"
          using old in1 allTyp unfolding subPrefixStructuralConsistency_def by fastforce
        then consider (val) v val' where "accessStore locs (Memory st) = Some (MValue v)
                                        ∧ MCon tp (Memory st) locs ∧ tp = MTValue val'
                                        ∧ accessTypeStore locs (Memory st) = Some tp"
          | (ptr) p len arr where "accessStore locs (Memory st) = Some (MPointer p) ∧
                                  MCon tp (Memory st) p ∧ tp = MTArray len arr ∧
                                  (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some
arr)"
          using notNo by (auto split:option.splits memoryvalue.splits)

        then show "case accessStore locs (Memory st') of None ⇒ False
      | Some (MValue v) ⇒
          ∃ val. MCon tp (Memory st') locs ∧ tp = MTValue val ∧ accessTypeStore locs (Memory st') = Some
tp
      | Some (MPointer p) ⇒
          ∃ len arr.
            MCon tp (Memory st') p ∧
            tp = MTArray len arr ∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some
arr)"
        proof(cases)
          case val
          then show ?thesis
          proof(cases "locs = l")
            case True
            then have "tp = (MTValue t')" using lInfo(6) allTyp in1 by simp
            moreover have "∃ v''. accessStore locs (Memory st') = Some (MValue v'')"
              using True mcl MCon_sub_MTVal_imps_val lsublocs mcPrntNew by presburger
            ultimately show ?thesis using mcl lInfo(6) allTyp val True by fastforce
          next
            case False
```

```
          then show ?thesis using val nonLocChanged allTyp by auto
        qed
      next
        case ptr
        then have notlLocs:"locs ≠ l"
          using mconlOld by force
        have cc1:"(∀i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some arr)"
          using ptr allTyp by simp
        have "MCon (MTArray len arr) (Memory st) p" using ptr by blast

        then have "MCon (MTArray len arr) (Memory st') p" using cc1
        proof(induction arr arbitrary:len p)
          case (MTArray x11 x12)
          have oldexp:"(∀i<len. ∃val.
      accessStore (hash p (ShowL_nat i)) (Memory st) = Some (MPointer val) ∧
          MCon (MTArray x11 x12) (Memory st) val)"
            using MTArray.prems(1) MCon.simps(2)[of len "MTArray x11 x12" "Memory st" p]
            by (meson CompMemType.simps(2) CompTypeRemainsMCon MConArrayPointers)


          then have subs:"∀i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some (MTArray
x11 x12)"
            using MTArray by simp

          have "(∀i<len. ∃val.
      accessStore (hash p (ShowL_nat i)) (Memory st') = Some (MPointer val) ∧
          (MCon (MTArray x11 x12) (Memory st') (val)))"
          proof(intros)
            fix i assume idef:"i<len"

            then show "∃val. accessStore (hash p (ShowL_nat i)) (Memory st') = Some (MPointer val) ∧
                       MCon (MTArray x11 x12) (Memory st') val"
            proof(cases "(hash p (ShowL_nat i)) = l")
              case True
              then show ?thesis using mconlOld oldexp idef by auto
            next
              case False
              then have sameACC:"accessStore (hash p (ShowL_nat i)) (Memory st) = accessStore (hash p
(ShowL_nat i)) (Memory st')"
                using 3(4)unfolding updateStore_def accessStore_def by simp
              then obtain val where mcc:"accessStore (hash p (ShowL_nat i)) (Memory st) = Some
(MPointer val)
                                    ∧ MCon (MTArray x11 x12) (Memory st) val"
                using oldexp unfolding MCon.simps using idef by blast
              then have "accessStore (hash p (ShowL_nat i)) (Memory st') = Some (MPointer val) "
                using sameACC by simp
              have "∀i<x11. accessTypeStore (hash val (ShowL_nat i)) (Memory st) = Some x12"
                using 2(1) subs allTyp idef mcc unfolding TypeSafe_def subPrefixStructuralConsistency_def
                by force
              then have "MCon (MTArray x11 x12) (Memory st') val"
                using MTArray.IH[of x11 val] mcc idef allTyp by simp
              then show ?thesis using mcc sameACC idef oldexp by simp
            qed
          qed

          moreover have "(∃p'. accessStore p (Memory st') = Some (MPointer p') ∨ accessStore p
(Memory st') = None)"
          proof -
            have "p ≠ l" using mconlOld MTArray.prems(1) MCon.simps(2)[of len _ "Memory st" p] by
auto
            then show ?thesis using nonLocChanged MTArray.prems(1) MCon.simps(2)[of len _ "Memory
st" p] by simp
          qed
          moreover have "0 < len" using MTArray.prems(1) MCon.simps(2)[of len _ "Memory st" p] by
```

```
simp
            ultimately show ?case using nonLocChanged MCon.simps(2)[of len _ "Memory st'" p] by auto
          next
            case (MTValue x2)
            have oldexp:"(∀ i<len. ∃ val.
       accessStore (hash p (ShowL_nat i)) (Memory st) = Some (MValue val) ∧
           (MCon (MTValue x2) (Memory st) (hash p (ShowL_nat i))))"
              using MTValue.prems(1) MCon.simps(2)[of len "MTValue x2" "Memory st" p]
              by (auto split:option.splits memoryvalue.splits mtypes.splits)

            then have subs:"∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some (MTValue
x2)"
              using MTValue by simp


            have "(∀ i<len. ∃ val.
       accessStore (hash p (ShowL_nat i)) (Memory st') = Some (MValue val) ∧
           (MCon (MTValue x2) (Memory st') (hash p (ShowL_nat i))))"
            proof(intros)
              fix i assume idef:"i<len"

              then show "∃ val. accessStore (hash p (ShowL_nat i)) (Memory st') = Some (MValue val) ∧
                             MCon (MTValue x2) (Memory st') (hash p (ShowL_nat i))"
              proof(cases "(hash p (ShowL_nat i)) = l")
                case True
                then have "(MTValue x2) = (MTValue t')" using subs lInfo(6) idef allTyp by auto
                then show ?thesis using 3(4) True
                  unfolding updateStore_def accessStore_def using idef mcl by simp
              next
                case False
                then have sameACC:"accessStore (hash p (ShowL_nat i)) (Memory st) = accessStore (hash p
(ShowL_nat i)) (Memory st')"
                  using 3(4)unfolding updateStore_def accessStore_def by simp
                then have "MCon (MTValue x2) (Memory st') (hash p (ShowL_nat i))"
                  using oldexp unfolding MCon.simps
                  using idef by force
                then show ?thesis using sameACC idef oldexp by metis
              qed
            qed

            moreover have "(∃ p'. accessStore p (Memory st') = Some (MPointer p') ∨ accessStore p
(Memory st') = None)"
            proof -
              have "p ≠ l" using mconlOld MTValue(1) MCon.simps(2)[of len "MTValue x2" "Memory st" p]
by auto
              then show ?thesis using nonLocChanged MTValue(1) MCon.simps(2)[of len "MTValue x2"
"Memory st" p] by simp
            qed
            moreover have "0 < len" using MTValue(1) MCon.simps(2)[of len "MTValue x2" "Memory st" p]
by simp
            ultimately show ?case using nonLocChanged MCon.simps(2)[of len "MTValue x2" "Memory st'"
p] by auto
          qed
          then show ?thesis using ptr nonLocChanged notlLocs allTyp by auto
        qed
      qed
    next
      show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
        using 3(4) unfolding updateStore_def accessTypeStore_def by simp
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs
(Memory st') = Some (MPointer v')"
        using 3(4)
```

```
      by (metis MConAccessSame.simps(1) memoryvalue.distinct(1) SameMCon_imps_MConAccessSame mconlOld
nonLocChanged option.inject)
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
        by (metis MCon_sub_MTVal_imps_val lInfo(7) mcPrntNew nonLocChanged)
    next
      show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
          (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        by (metis MCon_sub_MTVal_imps_val accessStore_def fmdom_notI fmlookup_dom_iff lsublocs
nonLocChanged)
    next
      have a12:"Toploc (Memory st) = Toploc (Memory st')" using 3 unfolding updateStore_def by auto
      then show "Toploc (Memory st) ≤ Toploc (Memory st')" by simp
    qed
  qed
qed
```

Assignment case 4: a calldata array assigned to a memory variable on the stack: the array is copied into fresh memory.

```
lemma TypeSafe_Assign_4:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KCDptr p, Calldata (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStackloc l, type.Memory t'),g')"
    and "(MTArray x t) =t'"
    and "accessStore l (Stack st) = Some (KMemptr p')"
    and "(updateStore l (KMemptr (ShowL_nat (Toploc (Memory st)))) (Stack st)) = st''"
    and "cpm2m p (ShowL_nat (Toploc (Memory st))) x t cd (snd (allocate (Memory st))) = Some m"
    and "st' = st(|state.Gas := g', Memory := m, Stack := st''|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 4 = assms(4-10)
  have nonChangedStack:"∀loc. loc ≠ l ⟶ accessStore loc (Stack st) = accessStore loc (Stack st')"
using 4 unfolding accessStore_def updateStore_def by auto
  have accessLStack:"accessStore l (Stack st') = Some (KMemptr (ShowL_nat (Toploc (Memory st))))" using
4 unfolding accessStore_def updateStore_def by auto
  have sameStorage:"Storage st' = Storage st " using 4 by auto
  have lInDen:"(type.Memory t', Stackloc l) |∈| fmran (Denvalue env)" using lexpStackloc_imps_inDen[of
lv env cd _ _ l ] 4(2) by simp
  have nonLocChanged:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs
(snd (allocate (Memory st))) = accessStore locs m"
    using 4(6) unfolding cpm2m_def using cpm2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t cd
" (snd
      (allocate (Memory st)))" x m ] cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd " (snd
(allocate
      (Memory st)))" ]
    by fastforce
  have a30:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
      ⟶ accessStore locs (snd (allocate (Memory st))) = accessStore locs m"
    using 4(6) unfolding cpm2m_def using cpm2mSingleChange_typedPref[of p "(ShowL_nat (Toploc (Memory
st)))" t cd " (snd (allocate (Memory st)))" x m ] by fastforce
  have selfPoint:"∀l l'. TypedMemSubPref l (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∧ accessStore
l m = Some (MPointer l') ⟶ l' = l"
    using 4(6) unfolding cpm2m_def using cpm2mSelfPointers[of p "(ShowL_nat (Toploc (Memory st)))" t cd
" (snd (allocate (Memory st)))" x m ] by blast
  have mInStd:"m = Memory st'" using 4 by simp

  have p'MCon:"MCon t' (Memory st) p'" using 2 4(4) unfolding TypeSafe_def typeCompat_def using lInDen
```

by *(auto split:denvalue.splits option.splits stackvalue.splits type.splits)*

have *MConsrc:"MCon (MTArray x t) cd (extractValueType (KCDptr p)) ∧ (∃xx. KCDptr p = KCDptr xx) ∧*
(∃stloc tp'' pa.
(Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
accessStore stloc (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some (KCDptr pa) ∧
(tp'' = (MTArray x t) ∧ KCDptr p = KCDptr pa ∨
(∃len arr. extractValueType (KCDptr p) ≠ pa ∧ tp'' = MTArray len arr ∧ CompMemType cd len arr (MTArray x t) pa (extractValueType (KCDptr p)))))"
using *2(1) 4(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)"*
"(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KCDptr p" "Calldata (MTArray x t)" g]
2(3) **unfolding** *fullyInitialised_def* **by** *(auto split:type.splits if_splits)*
have *"(∃p. accessStore p' (Memory st) = Some (MPointer p)) ∨ accessStore p' (Memory st) = None"* **using** *p'MCon 4(3)*
by *(metis MCon.simps(2))*
note *limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]*
note *limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]*
moreover have *allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate (Memory st)))"* **using** *allocateSameAccess* **by** *blast*
ultimately have *"accessStore (ShowL$_{nat}$ (Toploc (Memory st))) (snd (allocate (Memory st))) = None"* **using** *LSubPrefL2_def* **by** *auto*
then have *MCondest:" MCon (MTArray x t) m (ShowL$_{nat}$ (Toploc (Memory st)))"* **using** *4(6)* **unfolding** *cpm2m_def*
using *MCon_cpm2m[of p "(ShowL$_{nat}$ (Toploc (Memory st)))" t cd "(snd (allocate (Memory st)))" x m] MConsrc extractValueType.simps(2)* **by** *metis*
have *selfPoint2:"∀l l'. LSubPrefL2 l (ShowL$_{nat}$ (Toploc (Memory st))) ∧ accessStore l (Memory st') = Some (MPointer l') ⟶ l' = l"* **using** *selfPoint limitSt1 limitSt*
"4"(6) a30 cpm2m_iter_dest_unchanged allocateSameAccess cpm2m_def eq_imp_le hash_inequality not_Some_eq
by *(metis mInStd)*

have *stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue env)*
∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc (ShowL$_{nat}$ (Toploc (Memory st)))"
using *typeSafe_noDenElementOverToploc_mem[OF 2(1)]* **by** *auto*

show *?thesis* **unfolding** *TypeSafe_def StateInvariant_def*
proof *intros*
show *"AddressTypes (Accounts st')"* **using** *2(1) 4* **unfolding** *TypeSafe_def* **by** *simp*
next
show *"unique_locations (Denvalue env)"* **using** *2(1)* **unfolding** *TypeSafe_def* **by** *simp*
next
have *a0:" compPointers (Stack st) (Denvalue env)"* **using** *2(1)* **unfolding** *TypeSafe_def* **by** *blast*
show *"compPointers (Stack st') (Denvalue env)"*
using *compPointers_stack_update[OF a0 nonChangedStack accessLStack]* .
next
have *"Accounts st = Accounts st'"* **using** *4* **by** *simp*
then show *"safeContract (Accounts st') (Storage st')"* **using** *sameStorage* **using** *2(1)* **unfolding** *TypeSafe_def safeContract_def* **by** *auto*
next
show *"balanceTypes (Accounts st')"* **using** *4* **using** *2(1)* **unfolding** *TypeSafe_def* **by** *simp*
next
have *"envAddressesWellFormed env"* **using** *2(1)* **unfolding** *TypeSafe_def* **by** *simp*
then show *"addressFormat (Address env)"* **and** *"addressFormat (Sender env)"* **by** *simp+*
next
show *"svalueTypes (Svalue env)"* **using** *2(1)* **unfolding** *TypeSafe_def* **by** *simp*
next
have **:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL$_{nat}$ tloc) ⟶ accessStore loc (Stack st) = None) ∧*
(∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc (ShowL$_{nat}$ tloc))) "

```
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 4 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
    proof intros

      fix tloc loc
      assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
      then show "accessStore loc (Stack st') = None" using *
        by (metis "4"(4) nonChangedStack not_None_eq)
    next
      fix loc y
      assume h1:" accessStore loc (Stack st') = Some y"
      then show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
        by (metis "4"(4) nonChangedStack)
    qed
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    have a10:"Toploc (snd (allocate (Memory st))) = Toploc m" using cpm2mTopLocSame[of p "(ShowL_nat
(Toploc (Memory
       st)))" t cd "(snd (allocate (Memory st)))" x m] 4(6) mInStd unfolding cpm2m_def by fastforce
    have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp
    have a20:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (snd
(allocate (Memory st))) = accessStore locs m"
      using 4(6) unfolding cpm2m_def using cpm2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t
cd "(snd (allocate (Memory st)))" x m ]
       cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd "(snd (allocate (Memory st)))" ] by
fastforce
    then have a30: "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs
(Memory st) = accessStore locs m"
      by (metis allocateSameAccess)
    have st'IsM:"Memory st' = m" using 4 by simp
    have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
    show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
    proof intros
      fix tloc loc
      assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then have b20:"Toploc (Memory st) ≤ tloc" using a10 tloc st'IsM by simp
      have "¬LSubPrefL2 loc p'"
      proof(rule ccontr)
        assume c10:"¬ ¬ LSubPrefL2 loc p'"
        then have c20: "LSubPrefL2 loc p'" by simp
        then have c30:"LSubPrefL2 p' (ShowL_nat tloc)" using b10 MemLSubPrefTransitive by auto
        then show False
        proof -
          have "MCon t' (Memory st) p'" using 4(4) 2(1) unfolding TypeSafe_def typeCompat_def using
lInDen by fastforce
          then obtain x i where c40: "accessStore p' (Memory st) = Some x ∨ accessStore (hash p'
(ShowL_nat i)) (Memory st) = Some x" using MCon_imps_Some by blast
          then show ?thesis
          proof(cases "accessStore p' (Memory st) = Some x")
            case True
            then show ?thesis using lessThanSome_imps_Locs[of "(state.Memory st)" p' x tloc] c30 a15
b20 by simp
          next
            case False
            then have "accessStore (hash p' (ShowL_nat i)) (Memory st) = Some x" using c40 by simp
            then show ?thesis using lessThanSome_imps_Locs_hash c30 a15 b20 by fastforce
          qed
        qed
      qed
      then show "accessStore loc (Memory st') = None " using b10 b20 a20 a10 st'IsM a15 a30
```

```
      by (metis (no_types, lifting) LSubPrefL2_def MemLSubPrefTransitive antisym_conv2
hash_inequality hash_suffixes_associative hashesIntSame lessThanTopLocs_def order_less_le_trans tloc)

    next
      fix loc y
      assume "accessStore loc (Memory st') = Some y "
      then show "∃tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
        by (metis a10 a15 a30 lessThanTopLocs_def order_less_trans st'IsM tloc)
    qed
  next
    show ns:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"
      show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using accessLStack nonChangedStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) accessLStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.distinct(3) stackvalue.simps(17) option.inject)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) accessLStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.distinct(7) stackvalue.simps(18) option.inject)
        next
          case (KMemptr x3)
          then have "∃struct. t'' = type.Memory struct"
          proof(cases "x1 = l")
            case True
            have "accessStore l (Stack st) = Some (KMemptr p')" using 4(4) by simp
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) KMemptr True unfolding
TypeSafe_def typeCompat_def by (cases t'';force+)
          next
            case False
            then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')" using accessLStack
nonChangedStack by simp
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) KMemptr unfolding TypeSafe_def
typeCompat_def by (cases t'';force+)
          qed
          then obtain struct where structdef:"t'' = type.Memory struct" by blast


          have "MCon struct (Memory st') x3"
          proof(cases "x1 = l")
            case True
            then have "x3 = (ShowL_nat (Toploc (Memory st)))" using 4 KMemptr adef unfolding
accessStore_def updateStore_def by auto
            moreover have "(MTArray x t) = struct"
              by (metis 2(1) "4"(3) Stackloc True type.inject(3) fst_conv inDen lInDen snd_conv
TypeSafe_def
                  structdef unique_locations_def)
            ultimately show ?thesis using MCondest 4 Stackloc True inDen lInDen mInStd structdef by
metis
          next
            case False
```

```
          then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')" using accessLStack
nonChangedStack by simp
          then have "accessStore x1 (Stack st) = Some (KMemptr x3)" using adef KMemptr by simp
          then have mgh:"MCon struct (Memory st) x3" using inDen KMemptr Stackloc adef structdef
2(1) unfolding TypeSafe_def typeCompat_def by force
          then show ?thesis using originalMConStillMCon[OF mgh limitSt ] mInStd allocateSameAccess
a30 by (metis nonLocChanged)

        qed

        then show ?thesis using structdef KMemptr Stackloc adef inDen Stackloc 2(1) unfolding
TypeSafe_def by simp

      next
        case (KStoptr x4)
        then show ?thesis using Stackloc adef sameStorage inDen Stackloc KStoptr 2(1) accessLStack
nonChangedStack unfolding TypeSafe_def typeCompat_def
          apply(cases t'')
            apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(20) type.distinct(3) type.simps(17) lInDen old.prod.inject snd_eqD uniqueLocs)
           apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(20) type.distinct(7) type.simps(18) lInDen old.prod.inject snd_eqD uniqueLocs)

          apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.distinct(11) stackvalue.simps(20) type.simps(19) option.inject)

          by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(20)
type.distinct(11) type.simps(20) lInDen prod.inject snd_conv uniqueLocs)
      qed

      next
      case (Storeloc x2)
      then show ?thesis using sameStorage inDen 2(1) unfolding TypeSafe_def typeCompat_def by
(cases t''; force)
    qed
  qed

  next
  have accSame:"Accounts st'= Accounts st" using 4 by auto
  from 2(3)
  obtain c_fi ct_fi dud_fi where fi:"
       Type (Accounts st (Address env)) = Some (atype.Contract c_fi) ∧
       Contract env = c_fi ∧
       ep $$ c_fi = Some (ct_fi, dud_fi) ∧
       (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v, Storeloc
id))) ∧
       (∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
       (∀t'' l'' p''.
           (type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack st) =
Some (KStoptr p'') ⟶
           (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l'
p''))"
      unfolding fullyInitialised_def by blast
  have fiPtrs':"∀t'' l'' p''.
           (type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack st') =
Some (KStoptr p'') ⟶
           (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l'
p'')"
  proof (intro allI impI)
    fix t'' l'' p''
    assume in1:"(type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env)∧accessStore l'' (Stack
st') = Some (KStoptr p'')"
    then show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t''
l' p''"
```

```
          using 4 fi unfolding accessStore_def updateStore_def
          by (metis 2(1) type.distinct(11) accessStore_def lInDen nonChangedStack prod.inject snd_eqD
typeSafeUnique uniqueLocs)
    qed
    show "fullyInitialised env (Accounts st') (Stack st')"
        unfolding fullyInitialised_def using fi fiPtrs' accSame by metis

    next
    have a20':"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (snd
(allocate (Memory st))) = accessStore locs m"
        using 4(6) unfolding cpm2m_def using cpm2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t
cd "(snd (allocate (Memory st)))" x m ]
          cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd "(snd (allocate (Memory st)))" ] by
fastforce
    then have a30': "∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs
(Memory st) = accessStore locs m"
        by (metis allocateSameAccess)
    have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
    have topLocEq:"Toploc (snd (allocate (Memory st))) = Toploc (Memory st')"
        using cpm2mTopLocSame 4(6) cpm2m_def mInStd MCondest
        by (metis (no_types, lifting) MCon.simps(2))
    next
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"
        unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t2 l2 ptr_loc sub_loc
      assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env) ∧ accessStore l2 (Stack st') =
Some (KMemptr ptr_loc)"

      show "case t2 of MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory
st') = Some arr)

     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
      proof(cases "ptr_loc = (ShowL_nat (Toploc (Memory st)))")
        case False
        then have sameACC:"accessStore l2 (Stack st') = accessStore l2 (Stack st)"
            using nonChangedStack * 4(5) unfolding accessStore_def updateStore_def
            by (metis stackvalue.inject(3) accessLStack accessStore_def option.inject)
        then have mcOld:"MCon t2 (Memory st) ptr_loc" using * by (metis 2(1) sameMemTSafe)
        then have old:"(case t2 of MTArray len arr ⇒
(∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
            using sameACC 2(1) * unfolding TypeSafe_def denvalueTypeCorrectness_def by fastforce

        have inDenvalue:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env)" using * by simp
        have lims:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 (ShowL_nat
(Toploc (Memory st))) ptr_loc"
            using typeSafe_noDenElementOverToploc_mem[OF 2(1) inDenvalue] * sameACC
            using LSubPrefL2_def MemLSubPrefTransitive by metis

        then show ?thesis
        proof(cases t2)
          case (MTArray x11 x12)
          have nonLocChanged_TypedSafe:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                      ⟶ accessTypeStore loc (Memory st') = accessTypeStore loc
(Memory st)"
              using lims mInStd 4(6) unfolding cpm2m_def
              by (metis allocateTypeSameAccess cpm2mSingleChange_Typed cpm2m_def)
          have nonLocChanged_TypedSafe2:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                      ⟶ accessStore loc (Memory st') = accessStore loc (Memory st)"
              using lims mInStd 4(6) cpm2mSingleChange unfolding cpm2m_def
              by (metis allocateSameAccess cpm2m_def)
          have "∀ i. ¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) (ShowL_nat (Toploc (Memory st)))"
```

```
            using lims Mutual_NonSub_SpecificNonSub by blast
          then show ?thesis using old MTArray nonLocChanged_TypedSafe nonLocChanged_TypedSafe2 lims
by simp
          next
          case (MTValue x2)
          have "¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc (Memory st)))" using lims by simp
          then have "accessTypeStore ptr_loc (Memory st') = accessTypeStore ptr_loc (Memory st)"
              by (metis "4"(6) allocateTypeSameAccess cpm2mSingleChange_Typed cpm2m_def mInStd)
          then show ?thesis using MTValue old by simp
        qed
        next
        case True
        then have newLocation:"l2 = l ∧ t2 = (MTArray x t)"
            using lInDen accessLStack *
            by (metis 2(1) "4"(3) LSubPrefL2_def Pair_inject type.inject(3) nonChangedStack snd_eqD
typeSafeUnique typeSafe_noDenElementOverToploc_mem
                unique_locations_def)
        then have l2IsL:"l2 = l" using newLocation by simp
        then have conc0:"ptr_loc = (ShowL_nat (Toploc (Memory st)))" using * 4(5,7) by auto
        have conc1:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) (Memory
st') = Some t"
            using 4(6) unfolding cpm2m_def
            using cpm2m_TypeCompChangeIndexs[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate
(Memory st)))" x m]
              mInStd by simp
        have nonLocChanged_TypedSafe2:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ∨ loc =
(ShowL_nat (Toploc (Memory st)))
                                        ⟶ accessStore loc (Memory st') = accessStore loc (Memory st)"
            using mInStd 4(6) unfolding cpm2m_def
            by (metis allocateSameAccess cpm2mSingleChange cpm2m_def)
        then show ?thesis using conc1 conc0 newLocation
            using LSubPrefL2_def limitSt by fastforce

      qed
    qed
    next
    have old:"subPrefixStructuralConsistency (Memory st)"
        using 2(1) unfolding TypeSafe_def by simp
    have none:"accessStore (ShowL_nat (Toploc (Memory st))) (Memory st) = None"
        by (simp add: <accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None>
allocateSame)
    have lessT:"lessThanTopLocs (Memory st)" using 2(1) unfolding TypeSafe_def by simp
    have someV:"SomeValSomeTyp (Memory st)"
        using 2(1) unfolding TypeSafe_def by simp
    have nonCha:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t) ⟶
          accessStore locs (Memory st) = accessStore locs (Memory st')"
        using a30 allocateSame mInStd by presburger
    have index:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) (Memory st')
= Some t"
        using "4"(6) cpm2m_TypeCompChangeIndexs cpm2m_def mInStd by auto
    have subs:"∀ destl'.
   TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x t) ⟶
   (∃ stt. CompMemType (Memory st') x t stt (ShowL_nat (Toploc (Memory st))) destl' ∧
          (case stt of
           MTArray parent_len parent_arr ⇒
             ∀ i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) (Memory st') = Some parent_arr
           | MTValue pval ⇒ accessTypeStore destl' (Memory st') = Some (MTValue pval)))"
        using cpm2m_TypeCompChange "4"(6) cpm2m_def mInStd by simp
    have subT:"∀ locs.
     ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs = (ShowL_nat (Toploc (Memory st))) ⟶
     accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using "4"(6) mInStd unfolding cpm2m_def using cpm2mSingleChange_Typed[of p "(ShowL_nat (Toploc
(Memory st)))" t cd " (snd (allocate (Memory st)))" x m]
          allocateTypeSameAccess by metis
```

```
    have subT2:"∀locs.
     ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∨ locs = (ShowL_nat (Toploc
(Memory st))) ⟶
     accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using "4"(6) mInStd unfolding cpm2m_def using cpm2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t cd " (snd (allocate (Memory st)))" x m]
          allocateTypeSameAccess by metis
    show "subPrefixStructuralConsistency (Memory st')"
        using cpm2m_subPrefixPersist[OF old none someV subT lessT subT2 subs _ _ index _ nonCha]
MCondest mInStd selfPoint nonLocChanged
        by (metis (no_types, lifting) "4"(6) allocateSame cpm2mSingleChange cpm2m_def)
    next
    have old:"(∀locs. (∃t. accessStore locs (Memory st) = Some t) = (∃tt. accessTypeStore locs (Memory
st) = Some tt))"
        using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    have somesome:"∀destl'. TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x t)
        ⟶ (∃t. accessStore destl' m = Some t) = (∃tt. accessTypeStore destl' m = Some tt)"
        using 4(6) unfolding cpm2m_def using cpm2m_TypeCompChange_somesome[of p "(ShowL_nat (Toploc
(Memory st)))" t cd "(snd (allocate (Memory st)))" x m]
        by simp
    have a30T:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
        ⟶ accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using 4(6) unfolding cpm2m_def using cpm2mSingleChange_typedPref_typed[of p "(ShowL_nat (Toploc
(Memory st)))" t cd " (snd (allocate (Memory st)))" x m ]
          mInStd allocateTypeSameAccess by metis
    show "SomeValSomeTyp (Memory st')"unfolding SomeValSomeTyp_def
    proof intros
      fix locs
      show "(∃t. accessStore locs (Memory st') = Some t) = (∃tt. accessTypeStore locs (Memory st') =
Some tt) "
      proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)")
        case True
        then have "(∃t. accessStore locs m = Some t) = (∃tt. accessTypeStore locs m = Some tt)"
            using somesome by simp
        then show ?thesis using mInStd by simp
        next
        case False
        then have "accessStore locs (Memory st) = accessStore locs (Memory st')"
            using a30 mInStd allocateSameAccess by metis
        moreover have "accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
            using a30T False by simp
        ultimately show ?thesis using old by metis
      qed
    qed
    fix locs t'
    assume in1:"accessTypeStore locs (Memory st) = Some t'"
    show "accessTypeStore locs (Memory st') = Some t'"
    proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)")
      case True
      then show ?thesis using 2(1) in1 unfolding TypeSafe_def lessThanTopLocs_def
          by (metis Suc_n_not_le_n neg_MemLSubPrefL2_imps_TypedMemSubPref not_less_eq_eq old
option.distinct(1))
      next
      case False
      then show ?thesis using in1 4(6) unfolding cpm2m_def using cpm2mSingleChange_typedPref_typed[of
p "(ShowL_nat (Toploc (Memory st)))" t cd " (snd (allocate (Memory st)))" x m ]
          mInStd allocateTypeSameAccess by metis
    qed
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using allocateSame limitSt mInStd nonLocChanged by (metis dual_order.refl option.discI)
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
```

```
st') = Some (MValue v')"
        by (metis allocateSameAccess limitSt mInStd nonLocChanged option.distinct(1) order_eq_refl)
    next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"

        by (metis LSubPrefL2_def MemLSubPrefTransitive Read_Show_nat'_id allocateSame
hash_flatten_right hash_inequality mInStd nat_less_le nonLocChanged)
    next
    have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
    have topLocEq:"Toploc (snd (allocate (Memory st))) = Toploc (Memory st')"
        using cpm2mTopLocSame 4(6) cpm2m_def mInStd MCondest
        by (metis (no_types, lifting) MCon.simps(2))
    then show "Toploc (Memory st) ≤ Toploc (Memory st')" using tloc by simp
  qed
qed
```

Assignment case 5: a calldata array assigned to a storage variable reached through a stack pointer: the array is copied into storage.

```
lemma TypeSafe_Assign_5:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st⦇state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KCDptr p, Calldata (MTArray x t)), g)"
    and "lexp lv env cd (st⦇state.Gas := g⦈) g = Normal((LStackloc l, type.Storage t'),g')"
    and "accessStore l (Stack st) = Some (KStoptr p')"
    and "cps2mTypeCompatible t' (MTArray x t) "
    and "cpm2s p p' x t cd (Storage st (Address env)) = Some s"
    and "st' = st⦇state.Gas := g', Storage := (Storage st) (Address env := s)⦈"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 5 = assms(4-9)
  obtain t''' where t''Def:"t' = STArray x t''' " using 5(4)
    by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
  have sameStack:"(Stack st) = (Stack st')" using 5 unfolding accessStore_def updateStore_def by auto
  have sameMemory:"Memory st' = Memory st " using 5 by auto
  have lInDen:"(type.Storage t', Stackloc l) |∈| fmran (Denvalue env)"
    using lexpStackloc_imps_inDen[of lv env cd ] 5(2) by simp
  have nonLocChanged:"∀t' locs. ¬ LSubPrefL2 locs p' ∨ locs = p' ⟶ accessStorage t' locs (Storage st
(Address env)) = accessStorage t' locs s"
    using 5(5) unfolding cpm2s_def using cpm2sSingleChange[of p p' t cd "(Storage st (Address env))" x
s]
    by fastforce
  have a30:" ∀locs t' t''.
     cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray
x t') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs s"
    using 5(5) unfolding cpm2s_def using cpm2sSingleChange_typedPref[of p "p'" t cd "(Storage st
(Address env))" x s ] by simp
  then have a35:"∀locs t''. locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
    accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs (Storage st' (Address
env))"
    using 5 t''Def by auto
  have mInStd:"s = Storage st' (Address env)" using 5 by simp

  have p'MCon:"SCon t' p' (Storage st (Address env))" using 2 5 unfolding TypeSafe_def typeCompat_def
using lInDen
    by (auto split:denvalue.splits option.splits stackvalue.splits type.splits)
```

```
  have MConsrc:"MCon (MTArray x t) cd (extractValueType (KCDptr p)) ∧ (∃xx. KCDptr p = KCDptr xx) ∧
          (∃stloc tp'' pa.
          (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
          accessStore stloc (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some
(KCDptr pa) ∧
          (tp'' = (MTArray x t) ∧ KCDptr p = KCDptr pa ∨
           (∃len arr. extractValueType (KCDptr p) ≠ pa ∧ tp'' = MTArray len arr ∧ CompMemType cd len
arr (MTArray x t) pa (extractValueType (KCDptr p)))))"
    using 2(1) 5(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st⦈)"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KCDptr p" "Calldata (MTArray x t)" g]
      2(3) unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have SCondest:"SCon (STArray x t''') p' s" using 5 unfolding cpm2s_def
    using MCon_cpm2s[of p p' t cd "(Storage st (Address env))" x s t'''] t''Def MConsrc
extractValueType.simps(2) by metis
  then have SCondest2:"SCon (STArray x t''') p' (Storage st' (Address env))"
    by (simp add: mInStd)
  then have SCondest3:"SCon t' p' (Storage st' (Address env))"
    by (simp add: mInStd t''Def)

  have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                          ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto

  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 5 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
     (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2
= Stackloc l2'
         ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then show " if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "TypedStoSubpref stl2 stl1 tp1")
        case True
        then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref stl1 stl2 tp2")
          case True
          then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
        next
          case False
          then show ?thesis using f1 by simp
        qed
```

```
    qed
  qed
next
  have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp
  show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
  proof intros
    fix e ct dud i tp
    assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
             ep $$ Contract (e::environment) = Some (ct, dud) ∧
             ct $$ i = Some (Var tp)"
    show "SCon tp i (Storage st' (Address e))"
    proof (cases "Address e = Address env")
      case False
      then have sameAddr:"Storage st' (Address e) = Storage st (Address e)" using 5 by simp
      moreover have "SCon tp i (Storage st (Address e))"
        using * scOld 5 unfolding safeContract_def
        by fastforce
      ultimately show ?thesis by simp
    next
      case addrEq:True
      have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
        using * 5 by simp
      have epOld:"ep $$ Contract e = Some (ct, dud)"
        using * by simp
      have ctOld:"ct $$ i = Some (Var tp)"
        using * by simp
      have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
        using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
      have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
        using denI by (simp add: fmranI)
      have oldSConE:"SCon tp i (Storage st (Address e))"
        using scOld typedOld epOld ctOld unfolding safeContract_def by blast
      have oldSCon:"SCon tp i (Storage st (Address env))"
        using oldSConE addrEq by simp
      have cmpPtr:"(if TypedStoSubpref p' i tp then CompStoType tp t' i p'
                    else if TypedStoSubpref i p' t' then CompStoType t' tp p' i else True)"
        using 2(1) 5(3) inDenI lInDen t''Def unfolding TypeSafe_def compPointers_def by blast
      have newSCon:"SCon tp i (Storage st' (Address env))"
        using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
      show ?thesis using newSCon addrEq by simp
    qed
  qed
next
  show "balanceTypes (Accounts st')" using 5 using 2(1) unfolding TypeSafe_def by simp
next
  have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
  then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
next
  show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
next
  have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Stack st) = None) ∧
           (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
    using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  have **:"Toploc (Stack st) = Toploc (Stack st')" using 5 unfolding updateStore_def by auto
  show "lessThanTopLocs (Stack st')" using sameStack * ** unfolding lessThanTopLocs_def by auto
next
  show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
next
  show "lessThanTopLocs (Memory st')" using sameMemory unfolding lessThanTopLocs_def
    by (simp add: limitSt limitSt1)
next
  show tcN:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
```

```
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"

      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using sameStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(17)
)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(18))
        next
          case (KMemptr x3)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack sameMemory unfolding
TypeSafe_def typeCompat_def by (cases t''; fastforce)
        next
          case (KStoptr x4)
          then obtain struct where structDef: "t'' = type.Storage struct" using Stackloc adef inDen
Stackloc 2(1) sameStack unfolding TypeSafe_def typeCompat_def
            by (cases t''; fastforce)

          have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t'''), Stackloc l) |∈| fmran (Denvalue env) ∧
            (l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4) ∨ l' = Storeloc x4) ∧
            (accessStore l (Stack st) = Some (KStoptr p')) ⟶
            (if TypedStoSubpref p' x4 struct then CompStoType struct (STArray x t''') x4 p'
             else if TypedStoSubpref x4 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x4 else True))"
            using 2(1) 5(3) lInDen inDen Stackloc structDef KStoptr unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref p' x4 struct then CompStoType struct (STArray x
t''') x4 p'
           else if TypedStoSubpref x4 p' (STArray x t''') then CompStoType (STArray x t''') struct p' x4
else True)"
            using 5(3) lInDen inDen Stackloc structDef KStoptr t''Def
            by (simp add: adef sameStack)

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Stackloc adef inDen Stackloc 2(1) sameStack structDef KStoptr
            by fastforce

          have cmpStoPtr3:"(if TypedStoSubpref p' x4 struct then CompStoType struct t' x4 p'
            else if TypedStoSubpref x4 p' t' then CompStoType t' struct p' x4 else True)"
            using cmpStoPtr2 t''Def by blast
          have "SCon struct x4 (Storage st' (Address env))"
            using SCon_update_array_subloc_cases[OF cmpStoPtr3 SCondest2 SCondest3 SConx4Old t''Def
a35] .
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack KStoptr structDef by
simp
        qed
```

```
      next
        case (Storeloc x2)
        then obtain struct where structDef: "t'' = type.Storage struct" using Storeloc inDen 2(1)
sameStack unfolding TypeSafe_def typeCompat_def
          by (cases t''; fastforce)

        have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t'''), Stackloc l) |∈| fmran (Denvalue env) ∧
             l' = Storeloc x2) ∧
            (accessStore l (Stack st) = Some (KStoptr p')) ⟶
            (if TypedStoSubpref p' x2 struct then CompStoType struct (STArray x t''') x2 p'
             else if TypedStoSubpref x2 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x2 else True)"
          using 2(1) 5(3) lInDen inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by
blast

        then have cmpStoPtr2:"(if TypedStoSubpref p' x2 struct then CompStoType struct (STArray x t''')
x2 p'
             else if TypedStoSubpref x2 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x2 else True)"
          using 5(3) lInDen inDen Storeloc structDef t''Def by (simp add: sameStack)

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce

        have cmpStoPtr3:"(if TypedStoSubpref p' x2 struct then CompStoType struct t' x2 p'
          else if TypedStoSubpref x2 p' t' then CompStoType t' struct p' x2 else True)"
          using cmpStoPtr2 t''Def by fastforce
        have "SCon struct x2 (Storage st' (Address env))"
          using SCon_update_array_subloc_cases[OF cmpStoPtr3 SCondest2 SCondest3 SConx4Old t''Def a35] .
        then show ?thesis using Storeloc inDen 2(1) sameStack Storeloc structDef by simp
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
      using sameStack by auto

  next

    have accSame:"Accounts st'= Accounts st" using 5 by auto
    from 2(3)
    obtain c_fi ct_fi dud_fi where fi:"
         Type (Accounts st (Address env)) = Some (atype.Contract c_fi) ∧
         Contract env = c_fi ∧
         ep $$ c_fi = Some (ct_fi, dud_fi) ∧
         (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v, Storeloc
id))) ∧
         (∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
         (∀t'' l'' p''.
             (type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack st) =
Some (KStoptr p'') ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l'
p''))"
      unfolding fullyInitialised_def by blast
    have fiPtrs':"∀t'' l'' p''.
             (type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack st') =
Some (KStoptr p'') ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l'
p'')"
    proof (intro allI impI)
      fix t'' l'' p''
      assume in1:"(type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env)∧accessStore l'' (Stack
```

```
st') = Some (KStoptr p'')"
      then show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l' p''"
        using 5 fi unfolding accessStore_def updateStore_def
        using 2(1) type.distinct(11) accessStore_def lInDen prod.inject snd_eqD typeSafeUnique uniqueLocs
        by simp
    qed
    show "fullyInitialised env (Accounts st') (Stack st')"
      unfolding fullyInitialised_def using fi fiPtrs' accSame by metis
  next
    have cc0:"∀l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack st) = Some (KMemptr ptr_loc)"
      using 5(6) unfolding updateStore_def accessStore_def by auto
    have sameMem:"Memory st = Memory st'" using 5(6) by simp
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"
      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
       MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      then show "case t of
     MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        using sameMem by metis
    qed
  next
    have sameMem:"Memory st = Memory st'" using 5(6) by simp
    show "subPrefixStructuralConsistency (Memory st')"
      using 2(1) sameMem unfolding TypeSafe_def by simp
  next
    show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 5(6) by simp
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') = Some t "
      using 5(6) by auto
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory st') = Some (MPointer v')"
      using 5(6) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory st') = Some (MValue v')"
      using 5(6) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 5(6) by auto
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 5(6) by simp
  qed
qed
```

Assignment case 6: a calldata array assigned to a storage location: the array is copied into storage.

```
lemma TypeSafe_Assign_6:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st - costs (ASSIGN lv
```

```
        ex) env cd st) = Normal ((KCDptr p, Calldata (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStoreloc l, type.Storage t'),g')"
    and "cps2mTypeCompatible t' (MTArray x t)"
    and "cpm2s p l x t cd (Storage st (Address env)) = Some s"
    and "st' = st(|state.Gas := g', Storage := (Storage st) (Address env := s)|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 6 = assms(4-8)
  obtain t''' where t''Def:"t' = STArray x t''' " using 6(3)
    by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
  have sameStack:"(Stack st) = (Stack st')" using 6 unfolding accessStore_def updateStore_def by auto
  have sameMemory:"Memory st' = Memory st " using 6 by auto
  have nonLocChanged:"∀t' locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStorage t' locs (Storage st
(Address env)) = accessStorage t' locs s"
    using 6(4) unfolding cpm2s_def using cpm2sSingleChange[of p l t cd "(Storage st (Address env))" x
s]
    by fastforce
  have a30:" ∀locs t' t''.
     cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x
t') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs s"
    using 6(4) unfolding cpm2s_def using cpm2sSingleChange_typedPref[of p "l" t cd "(Storage st
(Address env))" x s ] by simp
  then have a35:"∀locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t''') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs (Storage st' (Address
env))"
    using 6 t''Def by auto
  have mInStd:"s = Storage st' (Address env)" using 6 by simp

  have MConsrc:"MCon (MTArray x t) cd (extractValueType (KCDptr p)) ∧ (∃xx. KCDptr p = KCDptr xx) ∧
         (∃stloc tp'' pa.
         (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
         accessStore stloc (Stack (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))) = Some
(KCDptr pa) ∧
         (tp'' = (MTArray x t) ∧ KCDptr p = KCDptr pa ∨
          (∃len arr. extractValueType (KCDptr p) ≠ pa ∧ tp'' = MTArray len arr ∧ CompMemType cd len
arr (MTArray x t) pa (extractValueType (KCDptr p)))))"
    using 2(1) 6(1) exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st|))"
       "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KCDptr p" "Calldata (MTArray x t)" g]
     2(3) unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have SCondest:"SCon (STArray x t''') l s" using 6 unfolding cpm2s_def
    using MCon_cpm2s[of p l t cd "(Storage st (Address env))" x s t'''] t''Def MConsrc
extractValueType.simps(2) by metis
  then have SCondest2:"SCon (STArray x t''') l (Storage st' (Address env))"
    by (simp add: mInStd)
  then have SCondest3:"SCon t' l (Storage st' (Address env))"
    by (simp add: mInStd t''Def)

  have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                          ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto
```

```
  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 6 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
     (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2
= Stackloc l2'
         ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then show " if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "TypedStoSubpref stl2 stl1 tp1")
        case True
        then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref stl1 stl2 tp2")
          case True
          then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
        next
          case False
          then show ?thesis using f1 by simp
        qed
      qed
    qed
  next
    have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp
    show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
    proof intros
      fix e ct dud i tp
      assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
               ep $$ Contract (e::environment) = Some (ct, dud) ∧
               ct $$ i = Some (Var tp)"
      show "SCon tp i (Storage st' (Address e))"
      proof (cases "Address e = Address env")
        case False
        then have sameAddr:"Storage st' (Address e) = Storage st (Address e)" using 6 by simp
        moreover have "SCon tp i (Storage st (Address e))"
          using * scOld 6 unfolding safeContract_def by fastforce
        ultimately show ?thesis by simp
      next
        case addrEq:True
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * 6 by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
          using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
        have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
          using denI by (simp add: fmranI)
        have oldSConE:"SCon tp i (Storage st (Address e))"
          using scOld typedOld epOld ctOld unfolding safeContract_def by blast
        have oldSCon:"SCon tp i (Storage st (Address env))"
          using oldSConE addrEq by simp
```

```
  have uniq:"unique_locations (Denvalue env)"
    using 2(1) typeSafeUnique by simp
  have epEnv:"ep $$ Contract env = Some (ct, dud)"
    using epOld addrEq
    using 2(3) fullyInitialised_def typedOld by auto
  show ?thesis
  proof(cases rule:lexpStorageG[OF 2(1) 6(2) 2(3)])
    case lInDen:1
    have inDenL:"(type.Storage t', Storeloc l) |∈| fmran (Denvalue env)"
      using lInDen by simp
    have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                  else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
      using 2(1) 6(3) inDenI inDenL unfolding TypeSafe_def compPointers_def by blast
    have newSCon:"SCon tp i (Storage st' (Address env))"
      using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
    show ?thesis using newSCon addrEq by simp
  next
    case sub2:(2 l''' t)
    have inDenL:"(type.Storage t, Storeloc l''') |∈| fmran (Denvalue env)" using sub2 by simp
    have relL:"TypedStoSubpref l l''' t ∧ CompStoType t t' l''' l" using sub2 by simp
    show ?thesis
    proof(cases "i = l'''")
      case True
      have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage t, Storeloc l''')"
        using uniqueLocs[OF uniq inDenI inDenL] True by simp
      then have tpEq:"tp = t" by simp
      have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                    else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
        using relL tpEq True by simp
      have newSCon:"SCon tp i (Storage st' (Address env))"
        using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
      show ?thesis using newSCon addrEq by simp
    next
      case False
      have ctL:"ct $$ l''' = Some (Var t)"
        using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenL] .
      have nPrnt1:"¬TypedStoSubpref i l''' t"
          and nPrnt2:"¬TypedStoSubpref l''' i tp"
        using methodVarsNoPref False epOld ctOld ctL by blast+
      have n1:"¬TypedStoSubpref l i tp"
        using NotRelatedPrnt_imps_notRelatedChild[OF nPrnt1 nPrnt2] relL by blast
      have n2:"¬TypedStoSubpref i l t'"
        using NotReachablePrnt_imps_notReachableChild nPrnt1 nPrnt2 relL by blast
      have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                    else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
        using n1 n2 by simp
      have newSCon:"SCon tp i (Storage st' (Address env))"
        using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
      show ?thesis using newSCon addrEq by simp
    qed
  next
    case sub3:(3 l' t l'')
    have inDenStk:"(type.Storage t, Stackloc l'') |∈| fmran (Denvalue env)" using sub3 by simp
    have ptrL:"accessStore l'' (Stack st) = Some (KStoptr l')" using sub3 by simp
    have rel0:"TypedStoSubpref l l' t ∧ CompStoType t t' l' l" using sub3 by simp
    obtain tprnt lprnt where
        inDenPr:"(type.Storage tprnt, Storeloc lprnt) |∈| fmran (Denvalue env)"
        and compPr:"CompStoType tprnt t lprnt l'"
      using fiPtr_parent_from_fullyInitialised[OF 2(3) inDenStk ptrL] by blast
    have compPr2:"CompStoType tprnt t' lprnt l"
      using compPr rel0 CompStoType_trns by blast
    have relPr:"TypedStoSubpref l lprnt tprnt ∧ CompStoType tprnt t' lprnt l"
      using compPr2 CompStoType_imps_TypedStoSubpref by blast
    show ?thesis
```

```
          proof(cases "i = lprnt")
            case True
            have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage tprnt, Storeloc lprnt)"
              using uniqueLocs[OF uniq inDenI inDenPr] True by simp
            then have tpEq:"tp = tprnt" by simp
            have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                         else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
              using compPr2 tpEq True
              by (metis True tpEq compPr2 relPr)
            have newSCon:"SCon tp i (Storage st' (Address env))"
              using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
            show ?thesis using newSCon addrEq by simp
          next
            case False
            have ctPr:"ct $$ lprnt = Some (Var tprnt)"
              using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenPr] .
            have nPrnt1:"¬TypedStoSubpref i lprnt tprnt"
                and nPrnt2:"¬TypedStoSubpref lprnt i tp"
              using methodVarsNoPref False epOld ctOld ctPr by blast+
            have n1:"¬TypedStoSubpref l i tp"
              using NotRelatedPrnt_imps_notRelatedChild[OF nPrnt1 nPrnt2] relPr by blast
            have n2:"¬TypedStoSubpref i l t'"
              using NotReachablePrnt_imps_notReachableChild nPrnt1 nPrnt2 relPr by blast
            have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                         else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
              using n1 n2 by simp
            have newSCon:"SCon tp i (Storage st' (Address env))"
              using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
            show ?thesis using newSCon addrEq by simp
          qed
        qed
      qed
    qed
  next
    show "balanceTypes (Accounts st')" using 6 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Stack st) = None) ∧
           (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 6 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" using sameStack * ** unfolding lessThanTopLocs_def by auto
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    show "lessThanTopLocs (Memory st')" using sameMemory unfolding lessThanTopLocs_def
      by (simp add: limitSt limitSt1)
  next
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"

      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
```

```
folding TypeSafe_def typeCompat_def using sameStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(17)
)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(18))
        next
          case (KMemptr x3)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack sameMemory unfolding
TypeSafe_def typeCompat_def by (cases t''; fastforce)
        next
          case (KStoptr x4)
          then obtain struct where structDef: "t'' = type.Storage struct" using Stackloc adef inDen
Stackloc 2(1) sameStack unfolding TypeSafe_def typeCompat_def
            by (cases t''; fastforce)

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases rule:lexpStorageG[OF 2(1) 6(2) 2(3)])
            case lInDen:1
            have cmpStoPtr:"(
              (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
              (type.Storage (STArray x t'''), Storeloc l) |∈| fmran (Denvalue env) ∧
               l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
               ⟶
              (if TypedStoSubpref l x4 struct then CompStoType struct (STArray x t''') x4 l
               else if TypedStoSubpref x4 l (STArray x t''') then CompStoType (STArray x t''') struct l
x4 else True)"
              using 2(1) 6(3) lInDen inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

            then have cmpStoPtr2:"(if TypedStoSubpref l x4 struct then CompStoType struct (STArray x
t''') x4 l
               else if TypedStoSubpref x4 l (STArray x t''') then CompStoType (STArray x t''') struct l
x4 else True)"
              using 6(3) lInDen inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

            have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
              by fastforce

            have cmpStoPtr3:"(if TypedStoSubpref l x4 struct then CompStoType struct t' x4 l
              else if TypedStoSubpref x4 l t' then CompStoType t' struct l x4 else True)"
              using cmpStoPtr2 t''Def by fastforce
            have "SCon struct x4 (Storage st' (Address env))"
              using SCon_update_array_subloc_cases[OF cmpStoPtr3 SCondest2 SCondest3 SConx4Old t''Def
a35] .
            then show ?thesis by simp
          next
            case sub2:(2 l''' t)
            have cmpStoPtr:"(
              (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
              (type.Storage t, Storeloc l''') |∈| fmran (Denvalue env) ∧
               l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
               ⟶
              (if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
               else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
```

```
            using 2(1) 6(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
            using 6(3) sub2 inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
            by fastforce
          have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x4 struct")
            case True
            then have "CompStoType struct t x4 l'''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t''') x4 l" using sub2
              using CompStoType_trns t''Def by blast
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x4 l''' t")
              case True
              then have "l''' ≠ x4" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x4")
                case t5:True
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                moreover have "CompStoType t t' l''' x4" using t5 sub2 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x4 ")
                  case True
                  then show ?thesis
                    using SCon_imps_sublocs SCondest3 by blast
                next
                  case False
                  then show ?thesis
                    using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t struct l'''
x4› a35 scl''' sub2(2) t''Def by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x4 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub2 by blast
              then have asm20:"¬ TypedStoSubpref x4 l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x4 ≠ l" using False sub2 by blast
              have k7:"∀ locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 sub2 asm10
asm20 by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x4" using False
sub2 t''Def asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
```

```
blast
            qed
          qed

          then show ?thesis by simp
        next
          case sub3:(3 l''' t l'''')
          have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage t, Stackloc l'''') |∈| fmran (Denvalue env) ∧
             l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4)) ∧ accessStore l''''
(Stack st) = Some (KStoptr l''')
              ⟶
            (if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
            using 2(1) 6(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
            using 6(3) sub3 inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
            by fastforce
          have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x4 struct")
            case True
            then have "CompStoType struct t x4 l'''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t''') x4 l" using sub3
              using CompStoType_trns t''Def by blast
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x4 l''' t")
              case True
              then have "l''' ≠ x4" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x4")
                case t5:True
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                moreover have "CompStoType t t' l''' x4" using t5 sub3 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x4 ")
                  case True
                  then show ?thesis
                    using SCon_imps_sublocs SCondest3 by blast
                next
                  case False
                  then show ?thesis
                    using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t struct l'''
```

```
x4› a35 scl''' sub3(3) t''Def by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x4 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
              then have asm20:"¬ TypedStoSubpref x4 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x4 ≠ l" using False sub3 by blast
              have k7:"∀ locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 sub3 asm10
asm20 by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x4" using False
sub3 t''Def asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
            qed
          qed
          then show ?thesis by simp
        qed
        then show ?thesis using Stackloc inDen 2(1) sameStack KStoptr adef structDef by simp
      qed
    next
      case (Storeloc x2)
      then obtain struct where structDef: "t'' = type.Storage struct" using Storeloc inDen 2(1)
sameStack unfolding TypeSafe_def typeCompat_def
        by (cases t''; fastforce)

      have "SCon struct x2 (Storage st' (Address env))"
      proof(cases rule:lexpStorageG[OF 2(1) 6(2) 2(3)])
        case lInDen:1
        have cmpStoPtr:"(
          (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
          (type.Storage (STArray x t'''), Storeloc l) |∈| fmran (Denvalue env) ∧
           l' = Storeloc x2)
           ⟶
          (if TypedStoSubpref l x2 struct then CompStoType struct (STArray x t''') x2 l
           else if TypedStoSubpref x2 l (STArray x t''') then CompStoType (STArray x t''') struct l x2
else True)"
          using 2(1) 6(3) lInDen inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by
blast

        then have cmpStoPtr2:"(if TypedStoSubpref l x2 struct then CompStoType struct (STArray x
t''') x2 l
           else if TypedStoSubpref x2 l (STArray x t''') then CompStoType (STArray x t''') struct l x2
else True)"
          using 6(3) lInDen inDen Storeloc structDef t''Def sameStack by simp

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce

        have cmpStoPtr3:"(if TypedStoSubpref l x2 struct then CompStoType struct t' x2 l
          else if TypedStoSubpref x2 l t' then CompStoType t' struct l x2 else True)"
          using cmpStoPtr2 t''Def by fastforce
        have "SCon struct x2 (Storage st' (Address env))"
          using SCon_update_array_subloc_cases[OF cmpStoPtr3 SCondest2 SCondest3 SConx4Old t''Def
a35] .
        then show ?thesis by simp
      next
        case sub2:(2 l''' t)
        have cmpStoPtr:"(
            (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
            (type.Storage t, Storeloc l''') |∈| fmran (Denvalue env) )
```

```
              ⟶
             (if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
              else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
           using 2(1) 6(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

         then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
              else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
           using 6(3) sub2 inDen Storeloc structDef t''Def sameStack by auto

         have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
           by fastforce
         have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

         have "SCon struct x2 (Storage st' (Address env))"
         proof(cases "TypedStoSubpref l''' x2 struct")
           case True
           then have "CompStoType struct t x2 l''''" using cmpStoPtr2 by simp
           then have "CompStoType struct (STArray x t''') x2 l" using sub2
             using CompStoType_trns t''Def by blast
           then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
             by blast
         next
           case f1:False
           then show ?thesis
           proof(cases "TypedStoSubpref x2 l''' t")
             case True
             then have "l''' ≠ x2" using f1
               using TypedStoSubpref_sameLoc by auto
             then show ?thesis
             proof(cases "l = x2")
               case t5:True
               then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
               moreover have "CompStoType t t' l''' x2" using t5 sub2 by auto
               ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                 by simp
               then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
             next
               case False
               then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
               then show ?thesis
               proof(cases "CompStoType t' struct l x2 ")
                 case True
                 then show ?thesis
                   using SCon_imps_sublocs SCondest3 by blast
               next
                 case False
                 then show ?thesis
                   using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t struct l'''
x2› a35 scl''' sub2(2) t''Def by blast
                 qed
               qed
             next
               case False
               then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub2 by blast
               then have asm20:"¬ TypedStoSubpref x2 l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
               have notSame:"x2 ≠ l" using False sub2 by blast
               have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub2 asm10 asm20
by blast
               have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x2" using False sub2
```

```
t''Def asm10 asm20 by blast
            show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
          qed
        qed

        then show ?thesis by simp
      next
        case sub3:(3 l''' t l'''')
        have cmpStoPtr:"(
            (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
            (type.Storage t, Stackloc l'''') |∈| fmran (Denvalue env) ∧
             accessStore l'''' (Stack st) = Some (KStoptr l'''))
             ⟶
            (if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
             else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
          using 2(1) 6(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
             else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
          using 6(3) sub3 inDen Storeloc structDef t''Def sameStack by auto

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce
        have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases "TypedStoSubpref l''' x2 struct")
          case True
          then have "CompStoType struct t x2 l''''" using cmpStoPtr2 by simp
          then have "CompStoType struct (STArray x t''') x2 l" using sub3
            using CompStoType_trns t''Def by blast
          then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref x2 l''' t")
            case True
            then have "l''' ≠ x2" using f1
              using TypedStoSubpref_sameLoc by auto
            then show ?thesis
            proof(cases "l = x2")
              case t5:True
              then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
              moreover have "CompStoType t t' l''' x2" using t5 sub3 by auto
              ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
            next
              case False
              then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
              then show ?thesis
              proof(cases "CompStoType t' struct l x2 ")
                case True
                then show ?thesis
                  using SCon_imps_sublocs SCondest3 by blast
              next
                case False
                then show ?thesis
                  using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t struct l'''
```

```
x2› a35 scl''' sub3(3) t''Def by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
              then have asm20:"¬ TypedStoSubpref x2 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x2 ≠ l" using False sub3 by blast
              have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub3 asm10 asm20
by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x2" using False sub3
t''Def asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
            qed
          qed
          then show ?thesis by simp
        qed


        then show ?thesis using Storeloc inDen 2(1) sameStack Storeloc structDef by simp
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd" using
sameStack by simp

  next
    have "Accounts st'= Accounts st" using 6 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 6 unfolding
fullyInitialised_def by auto
  next
    have ss:"Stack st = Stack st' ∧ Memory st = Memory st'" using 6 by simp
    have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
      using 6(5) unfolding updateStore_def accessStore_def by auto
    have sameMem:"Memory st = Memory st'" using 6(5) by simp
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"
      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
       MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      then show "case t of
     MTArray len arr ⇒ (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        using sameMem by metis
    qed
  next
    have sameMem:"Memory st = Memory st'" using 6(5) by simp
    show "subPrefixStructuralConsistency (Memory st')"
      using 2(1) sameMem unfolding TypeSafe_def by simp
  next
    show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 6(5) by auto
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
      using 6(5) by auto
  next
```

```
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 6(5) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 6(5) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
       (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 6(5) by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 6(5) by auto
  qed
qed
```

Assignment case 7: a calldata array assigned to a memory location: the array is copied into fresh memory and the location is updated to point to it.

```
lemma TypeSafe_Assign_7:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
       ex) env cd st) = Normal ((KCDptr p, Calldata (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LMemloc l, type.Memory t'),g')"
    and "(MTArray x t) = t'"
    and "cpm2m p (ShowL_nat (Toploc (Memory st))) x t cd (snd (allocate (Memory st))) = Some m"
    and "(updateStore l (MPointer (ShowL_nat (Toploc (Memory st)))) m) = m'"
    and "st' = st(|state.Gas := g', Memory := m'|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 7 = assms(4-9)
  have temp:"TypeSafe env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|))) (Memory (st(|Gas := g|)))
(Storage (st(|Gas := g|))) cd"
    using 2(1) by simp
  have ttt:"fullyInitialised env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|)))" using 2(3) unfold-
ing fullyInitialised_def by simp
  then show ?thesis
  proof(cases rule:lexpIndexMem[OF 7(2) temp ttt])
    case lInfo:(1 x21 x22 tp tParent l' l'' prnt len' arr' i)
    have nonChangedStack:"∀loc. loc ≠ l ⟶ accessStore loc (Stack st) = accessStore loc (Stack st')"
using 7 unfolding accessStore_def updateStore_def by auto
    have sameStack:"(Stack st') = Stack st" using 7 unfolding accessStore_def updateStore_def by auto
    have sameStorage:"Storage st' = Storage st " using 7 by auto
    have nonLocChanged:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
       (ShowL_nat (Toploc (Memory st)))⟶ accessStore locs (snd (allocate (Memory st))) = accessStore
       locs m"
      using 7(4) unfolding cpm2m_def using cpm2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t
cd " (snd
         (allocate (Memory st)))" x m ] cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd " (snd
(allocate
         (Memory st)))" ]
      by fastforce
    have a30:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
       ⟶ accessStore locs (snd (allocate (Memory st))) = accessStore locs m"
      using 7(4) unfolding cpm2m_def using cpm2mSingleChange_typedPref[of p "(ShowL_nat (Toploc (Memory
st)))" t cd " (snd (allocate (Memory st)))" x m ] by fastforce
    have a32:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t) ⟶
accessStore locs (Memory st) = accessStore locs m"
      by (metis a30 allocateSameAccess)
    have selfPoint:"∀l l'. TypedMemSubPref l (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∧
```

```
accessStore
       l m = Some (MPointer l') ⟶ l' = l ∧ l ≠ (ShowL_nat (Toploc (Memory st)))"
      using 7(4) unfolding cpm2m_def using cpm2mSelfPointers[of p "(ShowL_nat (Toploc (Memory st)))" t
cd " (snd (allocate (Memory st)))" x m ]
      by (metis LSubPrefL2_def hash_inequality hash_suffixes_associative TypedMemSubPref.simps(2)
typedPrefix_imp_SubPref)

    have sameTypeAccess:" ∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
        (ShowL_nat (Toploc (Memory st))) ⟶
     accessTypeStore locs (Memory st) = accessTypeStore locs m"
      using 7(4) unfolding cpm2m_def
      using cpm2mSingleChange_Typed[of p "(ShowL_nat (Toploc (Memory st)))" t cd " (snd (allocate (Memory
st)))" x m ]
       allocateTypeSameAccess by metis

    have sameTypeAccess2:" ∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
        ∨ locs = (ShowL_nat (Toploc (Memory st))) ⟶
     accessTypeStore locs (Memory st) = accessTypeStore locs m"
      using 7(4) unfolding cpm2m_def
      using cpm2mSingleChange_typedPref_typed[of p "(ShowL_nat (Toploc (Memory st)))" t cd " (snd
(allocate (Memory st)))" x m ]
       allocateTypeSameAccess by metis

    have mInStd:"m' = Memory st'" using 7 by simp
    have NonChangeM'm:"∀locs. locs ≠ l ⟶ accessStore locs m = accessStore locs m'"
      using 7(5) unfolding accessStore_def updateStore_def by auto
    have NonChangeM'mT:"∀locs. accessTypeStore locs m = accessTypeStore locs m'"
      using 7(5) unfolding accessStore_def updateStore_def accessTypeStore_def by auto
    then have nonLocChanged3:"∀locs. locs ≠l ∧ ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st)))
(MTArray x t)
                              ⟶ accessStore locs (Memory st) = accessStore locs (Memory st')"
      using a32 mInStd NonChangeM'm by auto

    have accessL:"accessStore l m' = Some (MPointer (ShowL_nat (Toploc (Memory st))))" using 7(5) un-
folding accessStore_def updateStore_def by auto

    have MConsrc:"MCon (MTArray x t) cd (extractValueType (KCDptr p)) ∧ (∃xx. KCDptr p = KCDptr xx) ∧
         (∃stloc tp'' pa.
         (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
         accessStore stloc (Stack (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))) = Some
(KCDptr pa) ∧
         (tp'' = (MTArray x t) ∧ KCDptr p = KCDptr pa ∨
          (∃len arr. extractValueType (KCDptr p) ≠ pa ∧ tp'' = MTArray len arr ∧ CompMemType cd len
arr (MTArray x t) pa (extractValueType (KCDptr p)))))"
      using 2(1) 7(1) exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st|))"
         "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KCDptr p" "Calldata (MTArray x t)" g]
      using 2(3) unfolding fullyInitialised_def
      by (auto split:type.splits if_splits )

    have limitSt1:"(∀loc y. accessStore loc (Memory st) = Some y ⟶ (∃tloc<Toploc (Memory st).
LSubPrefL2 loc (ShowL_nat tloc)))"
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have limitSt:"(∀tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None)"
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    moreover have allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))"
      using allocateSameAccess by blast
    ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
    then have MCondest:" MCon (MTArray x t) m (ShowL_nat (Toploc (Memory st)))" using 7(4) unfolding
cpm2m_def
      using MCon_cpm2m[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory st)))" x m]
```

```
      MConsrc extractValueType.simps(2) by metis
    have lIs:"∃x. accessStore l (Memory st) = Some x" using lInfo by (auto split:option.splits)
    then have lOld:"∃p. accessStore l (Memory st) = Some (MPointer p)" using lInfo 7 by force
    then have l_not_toploc_orSub:"¬LSubPrefL2 l (ShowL_nat (Toploc (Memory st)))"
      using limitSt by fastforce
    then have lOld2:"∃p. accessStore l m = Some (MPointer p)" using lOld
      by (simp add: allocateSame nonLocChanged)


    have tps:"∀destl'.
   TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x t) ⟶
   (∃stt. CompMemType m x t stt (ShowL_nat (Toploc (Memory st))) destl' ∧
        (case stt of MTArray parent_len parent_arr ⇒ ∀i<parent_len. accessTypeStore (hash destl'
(ShowL_nat i)) m = Some parent_arr
         | MTValue pval ⇒ accessTypeStore destl' m = Some (MTValue pval)))"
      using 7(4) unfolding cpm2m_def
      using cpm2m_TypeCompChange[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory
st)))" x m] by simp

    have tps2:"∀i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) m = Some t"
      using 7(4) unfolding cpm2m_def
      using cpm2m_TypeCompChangeIndexs[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate
(Memory st)))" x m]
      by blast

    have selfPoint2:"∀l1 l2. LSubPrefL2 l1 (ShowL_nat (Toploc (Memory st)))
                    ∧ accessStore l1 (Memory st') = Some (MPointer l2) ⟶ l2 = l1 ∧ l1 ≠ (ShowL_nat
(Toploc (Memory st)))"
      by (metis NonChangeM'm a30 allocateSame l_not_toploc_orSub limitSt linorder_le_cases mInStd
option.distinct(1)
          selfPoint)

    obtain len subT where tParentType:"tParent = MTArray len subT" using lInfo by blast
    then obtain p'' where lOrigin:"accessStore l (Memory (st)) = Some (MPointer p'')"
        and compType:"CompMemType (Memory (st⦇Gas := g⦈)) len subT (MTArray x t) l'' p''"
        and lsublocs:"l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = (MTArray x t) ∧ MCon (MTArray
len' arr') (Memory (st)) prnt"
        and lsublocs3:"(prnt = l'' ∧ len = len' ∧ arr' = subT ∨ CompMemType (Memory (st)) len subT
(MTArray len' arr') l'' prnt)"
      using lInfo 7(3) by force
    then have lsublocs2:" CompMemType (Memory (st)) len' arr' (MTArray x t) prnt p''"
      using "7"(3) CompMemType.simps(2) by blast

    then have bb9:"∀subT subloc. CompMemType (Memory (st)) len' arr' subT prnt subloc ∧ subloc = p''
                              ⟶ subT = (MTArray x t)"
      using CompMemTypeSameLocsSameType lsublocs by blast

    have mconPrnt:"MCon (MTArray len' (MTArray x t)) (Memory st) prnt" using lsublocs by auto
    have ldef:"l = hash prnt (ShowL_nat i) ∧ i < len'" using lsublocs by auto

    have t6:"(type.Memory tParent, Stackloc l') |∈| fmran (Denvalue env)" using lInfo by blast
    have t7:" MCon (MTArray len subT) (Memory (st)) l''" using lInfo
      using tParentType by auto
    then have t8:"¬ LSubPrefL2 l'' (ShowL_nat (Toploc (Memory st)))" using limitSt limitSt1
typeSafe_noDenElementOverToploc_mem[OF 2(1) t6] lInfo(3) by simp
    have comptype2:"CompMemType m len subT (MTArray x t) l'' p''" using
        cpm2mCompMemTypeOld_imps_CompMemType[of "(Memory (st⦇Gas := g⦈))" len subT "(MTArray x t)" l''
p'' "(ShowL_nat (Toploc (Memory st)))" m ]
        compType nonLocChanged limitSt t7 t8 allocateSame by auto
    have t10:"CompMemType (Memory st) len subT (MTArray x t) l'' p''"
      using
        cpm2mCompMemTypeOld_imps_CompMemType[of "(Memory (st⦇Gas := g⦈))" len subT "(MTArray x t)" l''
p'' "(ShowL_nat (Toploc (Memory st)))" m ]
        compType nonLocChanged limitSt t7 t8 allocateSame by auto
```

```
    then have mconlOld:"MCon (MTArray x t) (Memory st) p''" using 7(3) lInfo lOrigin by force

    have den:"(type.Memory (MTArray len subT), Stackloc l') |∈| fmran (Denvalue env)
           ∧ accessStore l' (Stack st) = Some (KMemptr l'')" using lInfo tParentType by simp
    have nonLocChanged_TypedSafe:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                        ⟶ accessTypeStore loc (Memory st') = accessTypeStore loc
(Memory st)"
    proof -
      have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
          (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st))) =
          accessTypeStore locs m"
        using 7(4,5,6) cpm2mSingleChange_Typed[of p "(ShowL_nat (Toploc (Memory st)))" t cd " (snd
(allocate (Memory st)))" x m ]
          cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd " (snd (allocate (Memory st)))" ] by
simp
      then have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
          (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st))) =
          accessTypeStore locs m'"
        using 7(6,5) unfolding accessTypeStore_def updateStore_def by force
      then show "∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore loc
(Memory st') = accessTypeStore loc (Memory st)"
        using allocateTypeSameAccess mInStd by metis
    qed


    have nonLocChanged2:"∀locs. locs ≠ l ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st)))
        ⟶ accessStore locs (Memory st) = accessStore locs (Memory st')"
      using 7 nonLocChanged LSubPrefL2_def NonChangeM'm mInStd allocateSame by metis
    have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                        ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
      using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto
    have MCondest2:" MCon (MTArray x t) (Memory st') (ShowL_nat (Toploc (Memory st))) " using 7 MCondest
    proof -
      have "¬ LSubPrefL2 (ShowL_nat (Toploc (Memory st))) l"
        using l_not_toploc_orSub MemLSubPrefTransitive LSubPrefL2_refl by blast
      moreover have "∀d. ¬ LSubPrefL2 d l ⟶ accessStore d m = accessStore d (Memory st')"
        using NonChangeM'm mInStd LSubPrefL2_refl by metis
      ultimately show ?thesis
        using MCon_mem_preserved_disjoint_update[of "(ShowL_nat (Toploc (Memory st)))" l "MTArray x t" m
"Memory st'"] selfPoint l_not_toploc_orSub MCondest by blast
    qed
    have bb:"∀locs tp. CompMemType (Memory st) len subT tp l'' locs ⟶ locs ≠ (ShowL_nat
        (Toploc (Memory st))) ∧ ¬LSubPrefL2 locs (ShowL_nat (Toploc (Memory st)))"
    proof intros
      fix locs tp
      assume asm1:"CompMemType (Memory st) len subT tp l'' locs"
      then have a2:"locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc
(Memory st)))" using t7
      proof(induction subT arbitrary:len l'')
        case (MTArray x1 subT)
        then show ?case
          using CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def limitSt limitSt1 subPtrs_nonTop by
blast
      next
        case (MTValue x)
        then show ?case
          by (metis (no_types, lifting) CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def limitSt
limitSt1 subPtrs_nonTop)
      qed
      then show " locs ≠ (ShowL_nat (Toploc (Memory st)))" by simp
      show "¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) " using a2 by simp
    qed
```

```
    have nonLocChanged22:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs = (ShowL_nat
(Toploc (Memory st)))
                          ⟶ accessStore locs (Memory st) = accessStore locs m" using nonLocChanged
      by (simp add: allocateSame)

    have prntMconNew:"MCon (MTArray len' (MTArray x t)) (Memory st') prnt"
    proof -
      have " ∀i<len'.
           (case accessStore (hash prnt (ShowL_nat i)) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x t of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x t) (Memory st') (hash prnt (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTArray x t of MTArray len' arr' ⇒ MCon (MTArray x t)
(Memory st') loc2 | MTValue Types ⇒ False))"
      proof intros
        fix i' assume asm1:"i'<len'"
        then obtain ptr where ptrDef': "accessStore (hash prnt (ShowL_nat i')) (Memory st) =
Some(MPointer ptr)
                                  " and ptrDef'2:"MCon (MTArray x t) (Memory st) ptr"
          using mconPrnt by (metis MConArrayPointers MCon_imps_sub_Mcon bot_nat_0.not_eq_extremum
not_less_zero)

        show "case accessStore (hash prnt (ShowL_nat i')) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x t of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x t) (Memory st') (hash prnt (ShowL_nat i')))
            | Some (MPointer loc2) ⇒ (case MTArray x t of MTArray len' arr' ⇒ MCon (MTArray x t)
(Memory st') loc2 | MTValue Types ⇒ False)"
        proof(cases "i' = i")
          case True
          then show ?thesis
            using MCondest2 accessL ldef mInStd by force
        next
          case False
          then have "accessStore (hash prnt (ShowL_nat i')) (Memory st) = accessStore (hash prnt
(ShowL_nat i')) (Memory st')" using ldef
            by (metis hashesIntSame limitSt nle_le nonLocChanged2 option.distinct(1) ptrDef')
          then have same:"accessStore (hash prnt (ShowL_nat i')) (Memory st') = Some(MPointer ptr)"
using ptrDef' by simp
          have notPrnt:"ptr ≠ prnt"
            using MConSubTypes mconPrnt ptrDef' CompMemJustType.simps(2) ptrDef'2 by blast

          have locRule:"∀ ct locs ints. CompMemType (Memory st) len' (MTArray x t) ct prnt locs ⟶
locs ≠
              prnt ∧ hash locs (ShowL_nat ints) ≠ l"
            using BothMConImpsNotCompMemType CompTypeRemainsMCon mconPrnt ldef ShowLNatDot
hash_injective by metis
          have "CompMemType (Memory st) len' (MTArray x t) (MTArray x t) prnt ptr" using ptrDef' asm1
by auto
          then have "MCon (MTArray x t) (Memory st') ptr" using ptrDef'2 locRule
          proof(induction t arbitrary: x ptr len' prnt)
            case (MTArray x11 x12)
            have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x11 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                 (MTArray x11 x12) (Memory st') (hash ptr (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x11 x12 of MTArray len' arr' ⇒ MCon (MTArray x11
x12) (Memory st') loc2 | MTValue Types ⇒ False))"
            proof intros
              fix i'' assume "i''<x"
              then obtain ptr' where ptr'Def:"accessStore (hash ptr (ShowL_nat i'')) (Memory st) =
Some(MPointer ptr')
                                          ∧ MCon (MTArray x11 x12) (Memory st) ptr'" using
MTArray.prems(2)
                by (metis MConArrayPointers MCon_imps_sub_Mcon neq0_conv not_less_zero)
              have "ptr ≠ prnt" using MTArray.prems by blast
```

```
            then have "(hash ptr (ShowL_nat i'')) ≠ l" using MTArray.prems ShowLNatDot hash_injective
by blast
            then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st) = accessStore (hash ptr
(ShowL_nat i'')) (Memory st')"
              by (metis less_or_eq_imp_le limitSt nonLocChanged2 option.discI ptr'Def)
            then have same2:"accessStore (hash ptr (ShowL_nat i'')) (Memory st') = Some(MPointer
ptr')" using ptr'Def by simp

            have "MCon (MTArray x11 x12) (Memory st) ptr'" using ptr'Def by blast
            moreover have "CompMemType (Memory st) x (MTArray x11 x12) (MTArray x11 x12) ptr ptr'"
using ptr'Def
              using ‹i'' < x› by auto
            moreover have "∀ct locs ints. CompMemType (Memory st) x (MTArray x11 x12) ct ptr locs
⟶ locs
                ≠ ptr ∧ hash locs (ShowL_nat ints) ≠ l"
              by (metis BothMConImpsNotCompMemType CompTypeRemainsMCon MTArray.prems(1,2,3)
compMemTypes_trns)
            ultimately have "MCon (MTArray x11 x12) (Memory st') ptr'" using MTArray.IH[of x x11 ptr
ptr'] by blast
            then show "(case accessStore (hash ptr (ShowL_nat i'')) (Memory st') of None ⇒ False
           | Some (MValue val) ⇒ (case MTArray x11 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
              (MTArray x11 x12) (Memory st') (hash ptr (ShowL_nat i'')))
           | Some (MPointer loc2) ⇒ (case MTArray x11 x12 of MTArray len' arr' ⇒ MCon (MTArray x11
x12) (Memory st') loc2 | MTValue Types ⇒ False))"
              using ptrDef' same2 by auto
          qed

          moreover have xNotZero:"x>0" using MTArray.prems(2)
            using bot_nat_0.not_eq_extremum by fastforce
          moreover have "(∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr
(Memory st') = None"
          proof(cases "ptr = l")
            case True
            then have " accessStore ptr (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))" using 7 by auto
            then show ?thesis by blast
          next
            case False
            then have pdef:"(∃p. accessStore ptr (Memory st) = Some (MPointer p)) ∨ accessStore ptr
(Memory st) = None"
              using MTArray.prems(2) MCon.simps(2)[of x _ "Memory st" ptr] xNotZero by simp
            then have "¬ LSubPrefL2 ptr (ShowL_nat (Toploc (Memory st)))" using limitSt1 limitSt
              by (metis (no_types, lifting) MCon_imps_Some MTArray.prems(2) LSubPrefL2_def
Not_Sub_More_Specific le_refl option.distinct(1))
            then have "accessStore ptr (Memory st) =accessStore ptr (Memory st') " using False
nonLocChanged2 by simp
            then show ?thesis using pdef by simp
          qed
          ultimately show ?case using MCon.simps(2)[of x "MTArray x11 x12" "Memory st'" ptr] by simp
        next
          case (MTValue x2)
          have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
             | Some (MValue val) ⇒ (case MTValue x2 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTValue x2) (Memory st') (hash ptr (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTValue x2 of MTArray len' arr' ⇒ MCon (MTValue x2)
(Memory st') loc2 | MTValue Types ⇒ False))"
          proof intros
            fix i'' assume "i''<x"
            then obtain val where oldDef:"accessStore (hash ptr (ShowL_nat i'')) (Memory st) =
Some(MValue val)
                                          ∧ MCon (MTValue x2) (Memory st) (hash ptr (ShowL_nat
i''))" using MTValue ptrDef'
              by (meson MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val)
            have "(hash ptr (ShowL_nat i'')) ≠ l " using MTValue ldef ShowLNatDot hash_injective by
```

```
blast
            then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st) = accessStore (hash ptr
(ShowL_nat i'')) (Memory st')"
              by (metis less_or_eq_imp_le limitSt nonLocChanged2 option.discI oldDef)
            then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st') = Some(MValue val)" using
oldDef by simp
            then show "case accessStore (hash ptr (ShowL_nat i'')) (Memory st') of None ⇒ False
                  | Some (MValue val) ⇒ (case MTValue x2 of MTArray l a ⇒ False | MTValue typ ⇒
                      MCon (MTValue x2) (Memory st') (hash ptr (ShowL_nat i'')))
                  | Some (MPointer loc2) ⇒ (case MTValue x2 of MTArray len' arr' ⇒ MCon (MTValue
x2) (Memory st') loc2 | MTValue Types ⇒ False)"
              using oldDef by auto
          qed
          moreover have xNotZero:"x>0" using MTValue(2)
            using bot_nat_0.not_eq_extremum by fastforce
          moreover have "(∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr
(Memory st') = None"
          proof(cases "ptr = l")
            case True
            then have " accessStore ptr (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))" using 7 by auto
            then show ?thesis by blast
          next
            case False
            then have pdef:"(∃p. accessStore ptr (Memory st) = Some (MPointer p)) ∨ accessStore ptr
(Memory st) = None"
              using MTValue(2) MCon.simps(2)[of x _ "Memory st" ptr] xNotZero by simp
            then have "¬ LSubPrefL2 ptr (ShowL_nat (Toploc (Memory st)))" using limitSt1 limitSt
              by (metis (no_types, lifting) MCon_imps_Some MTValue(2) LSubPrefL2_def
Not_Sub_More_Specific le_refl option.distinct(1))
            then have "accessStore ptr (Memory st) =accessStore ptr (Memory st') " using False
nonLocChanged2 by simp
            then show ?thesis using pdef by simp
          qed

          ultimately show ?case using MCon.simps(2)[of x "MTValue x2" "Memory st'" ptr] by simp
        qed
        then show ?thesis using same asm1 by simp
      qed
    qed

    moreover have "len' > 0" using mconPrnt
      using ldef by auto
    moreover have "prnt ≠ l" using ldef
      by (metis hash_inequality)
    moreover have "(∃p. accessStore prnt (Memory st') = Some (MPointer p)) ∨ accessStore prnt
(Memory st') = None"
    proof(cases "LSubPrefL2 prnt (ShowL_nat (Toploc (Memory st)))")
      case True
      then show ?thesis
        by (metis LSubPrefL2_def Not_Sub_More_Specific l_not_toploc_orSub ldef)
    next
      case False
      then show ?thesis using nonLocChanged2 calculation mconPrnt by simp
    qed
    ultimately show ?thesis using MCon.simps(2)[of len' "MTArray x t" "Memory st'" prnt] mconPrnt
nonLocChanged2 ldef by simp
  qed


  then have l''Top'':"CompMemType (Memory (st)) len subT (MTArray x t) l'' p''" using lsublocs3
lsublocs
    using compMemTypes_trns lsublocs2 by blast
```

```
    have aaa:"∀ sT sL. CompMemType (Memory (st)) len subT sT
        l'' sL ∧ sL ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 sL
            (ShowL_nat (Toploc (Memory st))) ∧ MCon (MTArray len subT) (Memory (st)) l''
                        ⟶ CompMemType m len subT sT l'' sL"
    proof(intros)
      fix sT sL
      assume asm1: " CompMemType (Memory st) len subT sT
          l'' sL ∧ sL ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 sL
              (ShowL_nat (Toploc (Memory st))) ∧ MCon (MTArray len subT) (Memory (st)) l'''
      then show " CompMemType m len subT sT l'' sL"
      proof(induction subT arbitrary: len l'')
        case (MTArray x1 subT)
        obtain i'' ptr where i''Def:"(i''<len ∧ accessStore (hash l'' (ShowL_nat i'')) (Memory st) =
Some
            (MPointer ptr) ∧ (ptr = sL ∧ MTArray x1 subT = sT ∨ CompMemType (Memory st) x1 subT sT ptr
sL))"
          using MTArray.prems unfolding CompMemType.simps by auto
        then have "accessStore (hash l'' (ShowL_nat i'')) (Memory st) = accessStore (hash l'' (ShowL_nat
i'')) m" using MTArray.prems
          by (metis Read_Show_nat'_id antisym_conv1 limitSt nonLocChanged22 option.distinct(1)
readLintNotEqual)

        then show ?case
        proof(cases "ptr = sL")
          case True
          then have "MTArray x1 subT = sT" using i''Def MTArray.prems
            by (meson BothMConImpsNotCompMemType CompTypeRemainsMCon MCon_imps_sub_Mcon)
          then show ?thesis using True i''Def
            using <accessStore (hash l'' (ShowL_nat i'')) (Memory st) = accessStore (hash l'' (ShowL_nat
i'')) m> by force
        next
          case False
          then have "CompMemType m x1 subT sT ptr sL" using MTArray.IH[of x1 ptr] using i''Def
MTArray.prems by force
          then show ?thesis
            by (metis <accessStore (hash l'' (ShowL_nat i'')) (Memory st) = accessStore (hash l''
                (ShowL_nat i'')) m> i''Def CompMemType.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case
          by auto
      qed
    qed


    have BT3:"∀ locs tp x t. ¬LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶
        ¬CompMemType (Memory (st')) x t tp (ShowL_nat (Toploc (Memory st))) locs
                                                                ∧ ¬TypedMemSubPrefPtrs (Memory (st'))
x t (ShowL_nat (Toploc (Memory st))) locs"
      using CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def SubPtrs_top selfPoint2 by blast

    have accessLGen:"∃ptr. accessStore l (Memory st') = Some (MPointer ptr)
                    ∧ LSubPrefL2 ptr (ShowL_nat (Toploc (Memory st))) ∧ MCon (MTArray x t) (Memory st')
ptr "
      using MCondest2 accessL mInStd
      by (simp add: LSubPrefL2_def )


    have notTopSublocs_inv:"∀ dloc1 x11 x12 stl1 i. ¬ LSubPrefL2 dloc1 (ShowL_nat (Toploc (Memory st)))
∧ TypedMemSubPrefPtrs (Memory st') x11 x12 stl1 dloc1
                        ⟶ TypedMemSubPrefPtrs (Memory st) x11 x12 stl1 dloc1"
    proof intros
```

```
      fix dloc1 x11 x12 stl1
      assume "¬ LSubPrefL2 dloc1 (ShowL_nat (Toploc (Memory st))) ∧ TypedMemSubPrefPtrs (Memory st') x11 x12 stl1 dloc1"
      then have asm3:"¬ LSubPrefL2 dloc1 (ShowL_nat (Toploc (Memory st)))"
          and asm4:"TypedMemSubPrefPtrs (Memory st') x11 x12 stl1 dloc1" by simp+
      show "TypedMemSubPrefPtrs (Memory st) x11 x12 stl1 dloc1 " using asm3 asm4
      proof(induction x12 arbitrary:x11 stl1)
        case (MTArray x1 x12)
        obtain i'' ptr where ptrDef:"i''<x11 ∧ accessStore (hash stl1 (ShowL_nat i'')) (Memory st') = Some (MPointer ptr)
                  ∧ (ptr = dloc1 ∨ TypedMemSubPrefPtrs (Memory st') x1 x12 ptr dloc1)"
          using MTArray.prems(2) unfolding TypedMemSubPrefPtrs.simps by auto
        then have notl:"(hash stl1 (ShowL_nat i'')) ≠ l" using MTArray.prems(1)
          by (metis BT3 LSubPrefL2_def memoryvalue.inject(2) accessL mInStd option.inject)
        have notSub:"¬LSubPrefL2 (hash stl1 (ShowL_nat i'')) (ShowL_nat (Toploc (Memory st)))"
          using SubPtrs_top asm3 ptrDef selfPoint2 by blast
        have sameAccess:"accessStore (hash stl1 (ShowL_nat i'')) (Memory st') = accessStore (hash stl1 (ShowL_nat i'')) (Memory st)"
          using notl notSub nonLocChanged2 by auto
        then show ?case
        proof(cases "ptr = dloc1")
          case True
          then show ?thesis using sameAccess ptrDef by force
        next
          case False
          then show ?thesis using MTArray.IH sameAccess ptrDef
            by (metis asm3 TypedMemSubPrefPtrs.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed

    have subPrefM:"subPrefixStructuralConsistency (m)"
      using cpm2m_subPrefixPersist[OF _ _ _ sameTypeAccess _ sameTypeAccess2 tps MCondest]
        ‹accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None› allocateSame
        nonLocChanged a32 tps2 selfPoint 2(1) unfolding TypeSafe_def by metis


    show ?thesis unfolding TypeSafe_def StateInvariant_def
    proof intros
      show "AddressTypes (Accounts st')" using 2(1) 7 unfolding TypeSafe_def by simp
    next
      show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
      then show "compPointers (Stack st') (Denvalue env)" using sameStack sameStorage by simp
    next
      show "safeContract (Accounts st') (Storage st')" using sameStorage 7 using 2(1) unfolding TypeSafe_def safeContract_def by auto
    next
      show "balanceTypes (Accounts st')" using 7 using 2(1) unfolding TypeSafe_def by simp
    next
      have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
      then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
    next
      show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have *:"((∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc (Stack st) = None) ∧
            (∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st). LSubPrefL2 loc (ShowL_nat tloc)))) "
        using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
```

```
      have **:"Toploc (Stack st) = Toploc (Stack st')" using 7 unfolding updateStore_def by auto
      show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
      proof intros

        fix tloc loc
        assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
        then show "accessStore loc (Stack st') = None" using *
          by (simp add: sameStack)
      next
        fix loc y
        assume h1:" accessStore loc (Stack st') = Some y"
        then show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
          by (metis sameStack)
      qed
    next
      show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
    next
      have a10:"Toploc (snd (allocate (Memory st))) = Toploc m"
        using cpm2mTopLocSame[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory st)))"
x m] 7(4) mInStd unfolding cpm2m_def by fastforce
      then have a12:"Toploc m = Toploc m'" using 7(5) unfolding updateStore_def by auto
      have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp
      have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
      show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have b20:"Toploc (Memory st) ≤ tloc" using a10 tloc a12
          using mInStd by force

        then show "accessStore loc (Memory st') = None "
          by (metis (no_types, lifting) LSubPrefL2_def MemLSubPrefTransitive NonChangeM'm <∃x.
accessStore l (Memory st) =
              Some x> a10 a12 allocateSame antisym_conv2 b10 hash_inequality
              hash_suffixes_associative hashesIntSame limitSt mInStd nonLocChanged option.discI
order_less_le_trans tloc)
      next
        fix loc y
        assume "accessStore loc (Memory st') = Some y "
        then show "∃tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
          by (metis NonChangeM'm <∃x. accessStore l (Memory st) = Some x> a10 a12 allocateSameAccess
limitSt1 mInStd nonLocChanged order_less_trans tloc)
      qed
    next
      show " typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
        unfolding typeCompat_def
      proof intros
        fix tLook lLook
        assume inDen:" (tLook, lLook) |∈| fmran (Denvalue env)"
        show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd tLook lLook"
        proof(cases lLook)
          case (Stackloc x1)
          then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1)
unfolding TypeSafe_def typeCompat_def using sameStack nonChangedStack by fastforce
          then show ?thesis
          proof(cases "a")
            case (KValue x1)
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
              by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(17) )
          next
```

```
            case (KCDptr x2)
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack nonChangedStack unfolding TypeSafe_def typeCompat_def
              by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(18))
          next
            case (KMemptr x3)
            then have "∃struct. tLook = type.Memory struct" using Stackloc adef inDen Stackloc 2(1) KMemptr sameStack unfolding TypeSafe_def typeCompat_def by (cases tLook;force+)
            then obtain struct where structdef:"tLook = type.Memory struct" by blast

            then have mcOld:"MCon struct (Memory st) x3" using 2(1) unfolding TypeSafe_def typeCompat_def
              using Stackloc adef inDen KMemptr sameStack by (auto split:type.splits denvalue.splits stackvalue.splits option.splits)
            then have mcM:"MCon struct m x3" using PreExistMconNotChangeByToploc limitSt limitSt1 nonLocChanged22
              by auto
            have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              using 2(1) inDen KMemptr Stackloc structdef adef 7(6)
              unfolding TypeSafe_def denvalueTypeCorrectness_def by simp
            then have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              by (metis (no_types, lifting))
            have nonLocChanged_TypedSafe:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                        ⟶ accessTypeStore loc m = accessTypeStore loc (Memory st)"
            proof -
              have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
                  (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st)))
                  = accessTypeStore locs m"
                using 7(4,5,6) cpm2mSingleChange_Typed[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory st)))" x m ]
                  cpm2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t cd " (snd (allocate (Memory st)))" ] by simp
              then show "∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore loc m = accessTypeStore loc (Memory st)"
                using allocateTypeSameAccess mInStd by metis
            qed

            then have "accessTypeStore x3 (Memory st) = accessTypeStore x3 m"
              using limitSt limitSt1 mcOld
              by (metis AllPtrsNotTop_MCon lessThanTopLocs_def nat_le_linear)
            moreover have eq2:"∀i::nat. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = accessTypeStore (hash x3 (ShowL_nat i)) (m)"
              using nonLocChanged_TypedSafe limitSt limitSt1 mcOld AllPtrsNotTop_MCon lessThanTopLocs_def
              by (metis MemLSubPrefL2_specific_imps_general nat_le_linear)
            ultimately have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) m = Some arr)
     | MTValue val ⇒ accessTypeStore x3 m = Some (MTValue val))" using structss by presburger

            have "accessTypeStore l (Memory st) = Some (MTArray x t)"
              using lInfo 7 nonLocChanged by auto
            then have accl:"accessTypeStore l (m) = Some (MTArray x t)"
              using nonLocChanged_TypedSafe
              using l_not_toploc_orSub by auto
            have "MCon struct (Memory st') x3"
              using cpm2m_singleLChange[OF mcM structss _ _ _ MCondest2 subPrefM] 7(5) mInStd
```

```
            by (metis NonChangeM'm accl lOld2)

          then show ?thesis using structdef KMemptr Stackloc adef inDen Stackloc 2(1) unfolding
TypeSafe_def typeCompat_def by simp

        next
          case (KStoptr x4)
          then show ?thesis using Stackloc adef sameStorage inDen Stackloc KStoptr 2(1)
nonChangedStack sameStack unfolding TypeSafe_def typeCompat_def
              apply(cases tLook) by fastforce+
        qed

        next
        case (Storeloc x2)
        then show ?thesis using sameStorage inDen 2(1) unfolding TypeSafe_def typeCompat_def by
(cases tLook; force)
       qed
     qed
     then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
using sameStack by simp
     next
     have "Accounts st'= Accounts st" using 7 by auto
     then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 7 unfolding
fullyInitialised_def by auto
     next
     have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
     have topLocEq:"Toploc (snd (allocate (Memory st))) = Toploc m"
         using cpm2mTopLocSame[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory
st)))" x m] 7(4) mInStd unfolding cpm2m_def by fastforce
     then have topmm':"Toploc m = Toploc m'" using 7(5) unfolding updateStore_def by auto
     then have topm'st':"Toploc m' = Toploc (Memory st')" using mInStd by blast

     then have "{l. Toploc (Memory st) ≤ l ∧ l < Suc (Toploc (Memory st))} = {(Toploc (Memory st))}"
by auto
     then have "{l. Toploc (Memory st) ≤ l ∧ l < (Toploc m)} = {(Toploc (Memory st))}" using
topLocEq mInStd unfolding allocate_def by auto
     then have topLocSet:"{(ShowL_nat l) |l. Toploc (Memory st) ≤ l ∧ l < Toploc m} = {ShowL_nat
(Toploc (Memory st))}"
         by (simp add: setcompr_eq_image)
     have "∀l. l ∈ WrittenMem_between (Memory st) (m) ⟶ TypedMemSubPref l (ShowL_nat (Toploc (Memory
st))) (MTArray x t) "
         using WrittenMem_between_def mInStd a30 selfPoint a32 by blast
     then have "∀l''. l'' ∈ WrittenMem_between (Memory st) (m') ⟶ TypedMemSubPref l'' (ShowL_nat
(Toploc (Memory st))) (MTArray x t) ∨ l'' = l"
         using NonChangeM'm WrittenMem_between_def by fastforce
     then have cc0:"∀l''. l'' ∈ WrittenMem_between (Memory st) (Memory st') ⟶ TypedMemSubPref
         l'' (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∨ l'' = l" using mInStd by simp
     moreover have aloccated:"∀x'. x'∈ (AllocatedMem_between (Memory st) m) ⟷ TypedMemSubPref x'
         (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∨ x' = (ShowL_nat (Toploc (Memory
         st)))"
     proof intros
       fix x'
       show "(x' ∈ AllocatedMem_between (Memory st) m) = (TypedMemSubPref x' (ShowL_nat (Toploc (Memory
st)))
           (MTArray x t) ∨ x' = (ShowL_nat (Toploc (Memory st))))"
       proof
         assume " x' ∈ AllocatedMem_between (Memory st) m"
         then have "x' ∈ {(ShowL_nat l) |l. Toploc (Memory st) ≤ l ∧ l < Toploc m} ∪ fset (fmdom
(Mapping m) |-| fmdom (Mapping (Memory st)))"
             unfolding AllocatedMem_between_def by simp
         then have "x' ∈ ({(ShowL_nat (Toploc (Memory st)))} ∪ fset (fmdom (Mapping m) |-| fmdom
(Mapping (Memory st))))"
             using topLocSet by simp
```

```
          then show "TypedMemSubPref x' (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∨ x' = (ShowL_nat
(Toploc (Memory st)))"
              using a30 accessStore_def allocateSame fminus_iff fmlookup_dom_iff
              by (metis Un_iff empty_iff insert_iff)
          next
          assume *:"TypedMemSubPref x' (ShowL_nat (Toploc (Memory st))) (MTArray x t) ∨ x' = (ShowL_nat
(Toploc (Memory st)))"

          then show "x' ∈ AllocatedMem_between (Memory st) m"
          proof(cases "x' = (ShowL_nat (Toploc (Memory st)))")
            case True
            then show ?thesis unfolding AllocatedMem_between_def using topLocSet by simp
            next
            case False
            then have "accessStore x' (Memory st) = None" using limitSt typedPrefix_imp_SubPref * by
blast

            moreover have "∃v. accessStore x' m = Some v" using * MCondest selfPoint
MCon_imps_TypedMemSubPref_Some False by blast

            ultimately have "x' ∈ fset (fmdom (Mapping m) |-| fmdom (Mapping (Memory st)))" using a30
                by (simp add: accessStore_def fmlookup_dom_iff)
            then show ?thesis using AllocatedMem_between_def by auto
          qed
        qed
      qed
      moreover have "accessStore l m = accessStore l (Memory st)"
          using l_not_toploc_orSub nonLocChanged22 by fastforce
      moreover obtain vv where vvdef:"accessStore l m = Some vv" using calculation lIs by auto
      moreover have lIn22:"l |∈| fmdom (Mapping m)" using fmdomI calculation(4) accessStore_def by
metis
      moreover have "accessStore l m' = Some (MPointer (ShowL_nat (Toploc (Memory st))))" using fmdomI
7(5) by auto
      moreover have "l |∈| fmdom (Mapping m')" using calculation(6) fmdomI
          by (metis accessStore_def)
      next
      show "denvalueTypeCorrectness env (Stack st') (Memory st')"
          unfolding denvalueTypeCorrectness_def
      proof intros
        fix t2 l2 ptr_loc
        assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env) ∧ accessStore l2 (Stack st')
= Some (KMemptr ptr_loc)"
        have sameACC:"accessStore l2 (Stack st') = accessStore l2 (Stack st)"
            using nonChangedStack * 7(5,6) unfolding accessStore_def updateStore_def by simp


        show "case t2 of MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i))
(Memory st') = Some arr)
                        | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        proof(cases t2)
          case (MTArray x11 x12)

          have old':"(∀i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12)"
              using sameACC 2(1) * unfolding TypeSafe_def denvalueTypeCorrectness_def using MTArray by
force

          have conc0:"(∀i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some x12)"
          proof -
            have o:"∀i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12" using
old' by blast
            then have "∀i<x11. ∃v. accessStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some v"
                using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by auto
            then have "∀i<x11. ¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) (ShowL_nat (Toploc (Memory
st)))"
```

```
            using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by fastforce
          then show ?thesis using nonLocChanged_TypedSafe o by simp
        qed

        then show ?thesis using MTArray by simp
        next
        case (MTValue x2)
        then show ?thesis using sameACC 2(1) * unfolding TypeSafe_def denvalueTypeCorrectness_def
            by (metis (no_types, lifting) mtypes.simps(6) nonLocChanged_TypedSafe stackDenvalLimits)
      qed
    qed
    next
    show "subPrefixStructuralConsistency (Memory st')"
        unfolding subPrefixStructuralConsistency_def
    proof intros
      fix locs tp
      assume in1:" accessTypeStore locs (Memory st') = Some tp "
      have sameTy:"∀ locs. accessTypeStore locs m = accessTypeStore locs (Memory st') "
          using mInStd by (metis NonChangeM'mT)
      then have in2:"accessTypeStore locs m = Some tp " using in1 by simp
      show "case accessStore locs (Memory st') of None ⇒ False
            | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st') locs ∧ tp = MTValue val ∧
accessTypeStore locs (Memory st') = Some tp
            | Some (MPointer p) ⇒
               ∃ len arr.
                  MCon tp (Memory st') p ∧
                  tp = MTArray len arr ∧
                  (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some arr)"
      proof(cases "locs = l")
        case True
        have "accessTypeStore l (Memory st) = Some (MTArray x t)"
            using lInfo 7 nonLocChanged by auto
        then have accl:"accessTypeStore l (m) = Some (MTArray x t)"
            using nonLocChanged_TypedSafe l_not_toploc_orSub
            using sameTypeAccess by auto
        then have tpDef:"tp = MTArray x t"
            using sameTy True in1 by simp
        then have ptr:"accessStore locs (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))"
            using mInStd True by (metis accessL)
        have sameAccm:"accessStore l m = accessStore l (Memory st)"
            using l_not_toploc_orSub nonLocChanged22 by fastforce
        moreover have " MCon tp (Memory st') (ShowL_nat (Toploc (Memory st)))"
            using MCondest2 tpDef by blast
        moreover have "(∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i))
(Memory st') = Some t)"
            using tpDef ptr subPrefM sameAccm accl unfolding subPrefixStructuralConsistency_def
            using sameTy tps2 by presburger
        moreover have "
        (∀ v. accessStore (ShowL_nat (Toploc (Memory st))) (Memory st') = Some (MPointer v)
              ⟶ accessTypeStore (ShowL_nat (Toploc (Memory st))) (Memory st') = Some (MTArray x t))"
            using LSubPrefL2_def selfPoint2 by blast
        ultimately show ?thesis using True tpDef ptr by auto
        next
        case False
        then have sameACC:"accessStore locs m = accessStore locs (Memory st')"
            using mInStd NonChangeM'm by simp
        then have in3:"(case accessStore locs m of None ⇒ False | Some (MValue v) ⇒ ∃ val. MCon tp
m locs
            ∧ tp = MTValue val ∧ accessTypeStore locs m = Some tp
     | Some (MPointer p) ⇒
        ∃ len arr.
           MCon tp m p ∧
           tp = MTArray len arr ∧
```

```
            (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) m = Some arr) )"
             using subPrefM in2 unfolding subPrefixStructuralConsistency_def by blast
          then show ?thesis
          proof(cases "accessStore locs m")
            case None
            then show ?thesis using in3 sameACC by simp
            next
            case (Some a)
            then show ?thesis
            proof(cases a)
              case (MValue x1)
              then obtain val where " MCon tp m locs ∧ tp = MTValue val ∧ accessTypeStore locs m =
Some tp"
                  using in3 Some by auto
              moreover have "MCon tp (Memory st') locs" using sameACC calculation by auto
              ultimately show ?thesis using sameACC Some MValue sameTy
                  by (auto split:option.splits)
              next
              case (MPointer x2)
              have "accessTypeStore l (Memory st) = Some (MTArray x t)"
                  using lInfo 7 nonLocChanged by auto
              then have accl:"accessTypeStore l (m) = Some (MTArray x t)"
                  using nonLocChanged_TypedSafe l_not_toploc_orSub
                  using sameTypeAccess by auto
              then obtain len arr where s:"MCon tp m x2 ∧ tp = MTArray len arr ∧
                        (∀ i<len. accessTypeStore (hash x2 (ShowL_nat i)) m = Some arr)"
                  using Some in3 MPointer by auto
              then have mcM:"MCon tp m x2" by auto
              then have "MCon tp (Memory st') x2"
                  using cpm2m_singleLChange[OF mcM _ _ _ _ MCondest2 subPrefM, of l]
                    7(5) accl mInStd s NonChangeM'm accessL lOld2
                  by simp
              moreover have "(∀ i<len. accessTypeStore (hash x2 (ShowL_nat i)) (Memory st') = Some arr)"
                  using s sameTy by simp

              ultimately show ?thesis using Some MPointer sameACC sameTy s
                  by (auto split:option.splits)
            qed
          qed
        qed
      qed

      next
      have old:"(∀ locs. (∃ t. accessStore locs (Memory st) = Some t) = (∃ tt. accessTypeStore locs
(Memory st) = Some tt))"
          using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
      have somesome:"∀ destl'. TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x t)
            ⟶ (∃ t. accessStore destl' m = Some t) = (∃ tt. accessTypeStore destl' m = Some tt)"
          using 7(4) unfolding cpm2m_def using cpm2m_TypeCompChange_somesome[of p "(ShowL_nat (Toploc
(Memory st)))" t cd "(snd (allocate (Memory st)))" x m]
          by simp
      have a30T:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
              ⟶ accessTypeStore locs (Memory st) = accessTypeStore locs m"
          using 7(4) unfolding cpm2m_def using cpm2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t cd " (snd (allocate (Memory st)))" x m ]
            mInStd allocateTypeSameAccess by metis
      show "SomeValSomeTyp (Memory st')"unfolding SomeValSomeTyp_def
      proof intros
        fix locs
        show "(∃ t. accessStore locs (Memory st') = Some t) = (∃ tt. accessTypeStore locs (Memory st') =
Some tt) "
        proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)")
          case True
          then have ss:"(∃ t. accessStore locs m = Some t) = (∃ tt. accessTypeStore locs m = Some tt)"
```

```
            using somesome by simp
          then show ?thesis
          proof(cases "locs = l")
            case True
            then show ?thesis using lInfo NonChangeM'mT accessL mInStd ss
                using lIs l_not_toploc_orSub nonLocChanged22 by auto
            next
            case False
            then show ?thesis using ss NonChangeM'mT NonChangeM'm mInStd by simp
          qed
          next
          case False
          then have acc1:"accessStore locs (Memory st) = accessStore locs m"
              using a30 mInStd allocateSameAccess by metis
          have acc2:"accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
              using a30T False mInStd NonChangeM'mT by simp
          show ?thesis
          proof(cases "locs = l")
            case True
            then show ?thesis using lInfo acc2 accessL mInStd by force
            next
            case False
            then have "accessStore locs m = accessStore locs (Memory st')"
                using mInStd NonChangeM'm by auto
            then show ?thesis using old acc1 acc2 by metis
          qed
        qed
      qed
      next
      fix locs t2
      assume acc1:"accessTypeStore locs (Memory st) = Some t2"
      then have someSome:"∃v. accessStore locs (Memory st) = Some v"
          using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by simp
      have a30T:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)
                ⟶ accessTypeStore locs (Memory st) = accessTypeStore locs m"
          using 7(4) unfolding cpm2m_def using cpm2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t cd " (snd (allocate (Memory st)))" x m ]
            mInStd allocateTypeSameAccess by metis

      show "accessTypeStore locs (Memory st') = Some t2"
      proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x t)")
        case True
        then show ?thesis
        proof(cases "locs = l")
          case True
          then show ?thesis using l_not_toploc_orSub
              by (metis NonChangeM'mT a30T acc1 mInStd typedPrefix_imp_SubPref)
          next
          case False
          then show ?thesis using NonChangeM'mT a30T typedPrefix_imp_SubPref acc1 NonChangeM'm mInStd
someSome
              by (metis less_or_eq_imp_le limitSt option.distinct(1))
        qed
        next
        case False
        have acc2:"accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
            using a30T False mInStd NonChangeM'mT by simp
        show ?thesis
        proof(cases "locs = l")
          case True
          then show ?thesis using lInfo acc2 accessL mInStd acc1 by argo
          next
          case False
          then have "accessStore locs m = accessStore locs (Memory st')"
```

```
            using mInStd NonChangeM'm by auto
          then show ?thesis using acc1 acc2 by metis
        qed
      qed
      next
      show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs
(Memory st') = Some (MPointer v')"
          by (metis accessL le_refl limitSt mInStd nonLocChanged2 option.distinct(1))
      next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
          by (metis memoryvalue.distinct(1) lOld le_refl limitSt nonLocChanged2 not_None_eq
option.inject)
      next
      fix i loc
      assume a1:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)"
          and a2:"accessStore loc (Memory st) = None"
      then have "loc ≠ l" using lOld by auto
      moreover have "¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))" using a1
          by (metis (no_types, opaque_lifting) LSubPrefL2_def MemLSubPrefTransitive hash_flatten_right
hash_inequality hashesIntSame nless_le)
      ultimately show " accessStore loc (Memory st') = None" using a2 nonLocChanged2 by simp
      next
      have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
      moreover have topLocEq:"Toploc (snd (allocate (Memory st))) = Toploc m"
          using cpm2mTopLocSame[of p "(ShowL_nat (Toploc (Memory st)))" t cd "(snd (allocate (Memory
st)))" x m] 7(4) mInStd unfolding cpm2m_def by fastforce
      moreover have topmm':"Toploc m = Toploc m'" using 7(5) unfolding updateStore_def by auto
      moreover have topm'st':"Toploc m' = Toploc (Memory st')" using mInStd by blast
      ultimately show "Toploc (Memory st) ≤ Toploc (Memory st')" by simp
    qed
  qed
qed
```

Assignment case 8: a memory array assigned to a memory variable on the stack: the stack now points to the same array.

```
lemma TypeSafe_Assign_8:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KMemptr p, type.Memory (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStackloc l, type.Memory t'),g')"
    and "t' = MTArray x t"
    and "st' = st(|state.Gas := g', Stack := updateStore l (KMemptr p) (Stack st)|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 8 = assms(4-7)
  then have stackChanged:"∀l'. l' ≠ l ⟶ accessStore l' (Stack st) = accessStore l' (Stack st')"
unfolding updateStore_def accessStore_def by simp
  have t'InDen:"(type.Memory t', Stackloc l) |∈| fmran (Denvalue env)" using lexpStackloc_imps_inDen[OF
8(2)] by blast
  then have lDen:"∀t'. (t', Stackloc l) |∈| fmran (Denvalue env) ⟶ t'= (type.Memory (MTArray x t))"
using 8 2(1) unfolding TypeSafe_def unique_locations_def by auto

  have pOrigin:"MCon (MTArray x t) (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)))
(extractValueType (KMemptr p)) ∧
      (∃xx. KMemptr p = KMemptr xx) ∧
      (∃stloc tp'' pa.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
```

```
          accessStore stloc (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some
(KMemptr pa) ∧
          (tp'' = (MTArray x t) ∧ KMemptr p = KMemptr pa ∨
           (∃len arr.
               extractValueType (KMemptr p) ≠ pa ∧
               tp'' = MTArray len arr ∧ CompMemType (Memory (st⦇Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st⦈)) len arr (MTArray x t) pa (extractValueType (KMemptr p)))))"
    using 2(1) 8(1) 2(3) using
      exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)"
"(state.Gas st - costs
          (ASSIGN lv ex) env cd st)" "KMemptr p" "type.Memory (MTArray x t)" g]
    by (auto split:type.splits if_splits )
  then obtain pParent pParentT pParentPtr where
    pOrigin:"MCon (MTArray x t) (Memory (st)) (extractValueType (KMemptr p)) ∧(type.Memory pParentT,
Stackloc pParent) |∈| fmran (Denvalue env) ∧
          accessStore pParent (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some
(KMemptr pParentPtr) ∧
          (pParentT = (MTArray x t) ∧ KMemptr p = KMemptr pParentPtr ∨
           (∃len arr.
               extractValueType (KMemptr p) ≠ pParentPtr ∧
               pParentT = MTArray len arr ∧ CompMemType (Memory (st⦇Gas := state.Gas st - costs (ASSIGN
lv ex) env cd
                    st⦈)) len arr (MTArray x t) pParentPtr (extractValueType (KMemptr p))))"
    using pOrigin by force
  have pMCon:"MCon (MTArray x t) (Memory st) p" using pOrigin by force

  have storageSame:"(Storage st' (Address env)) = (Storage st (Address env))" using 8 by simp
  have memorySame:"Memory st = Memory st'" using 8 by simp
  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof (intros)
    show tcN:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof(intros)
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t l'"
      proof (split denvalue.split, intros)
        fix loc assume a20:"l' = Stackloc loc"
        show "(case accessStore loc (Stack st') of None ⇒ False
              | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
              | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
              | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc
| _ ⇒ False)
              | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False))"
        proof (cases "loc = 1")
          case False
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then have "accessStore loc (Stack st) = None" using 8(4) False by (simp
add:stackSingleUpdate)
            then show False using a30 a20 a10 assms TypeSafe_def False 2(1)
              by (metis option.distinct(1) typeSafeLocExists)
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have a40:"accessStore loc (Stack st) = Some x2" using 8(4) False by (simp
add:stackSingleUpdate)
            then have a50:"(Memory st) = (Memory st')" using 8(4) by simp
            then have a60:"(Storage st) = (Storage st')" using 8(4) by simp
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
                  | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
                  | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc |
_ ⇒ False)
                  | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
```

```
(Address env)) | _ ⇒ False)"
            using a10 a20 a30 a40 a50 a60 2(1) unfolding TypeSafe_def typeCompat_def by (cases x2;
cases t; force)
          qed
        next
          case sameLoc:True
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then show False using a20 a10 assms(1) sameLoc 8(4) notNoneUpdate[of st' g' loc "(KMemptr
p)" ] by simp
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have x2IsP:"x2 = KMemptr p" using 8 sameLoc by auto
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
                  | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
                  | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc |
_ ⇒ False)
                  | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
            proof(cases "x2")
              case (KValue x1)

              then show ?thesis using x2IsP by auto
            next
              case (KCDptr x2)
              then show ?thesis using x2IsP by (simp add:sameLoc)
            next
              case (KMemptr x3)
              then show ?thesis
                using a10 a20 lDen memorySame pMCon sameLoc x2IsP by fastforce
            next
              case (KStoptr x4)
              then show ?thesis using x2IsP by (simp add:sameLoc)
            qed
          qed
        qed
      next
        fix x2 assume a20:"l' = Storeloc x2"
        then have "(Storage st' (Address env)) = (Storage st (Address env))" using 8(4) by simp
        then show "case t of type.Storage typ ⇒ SCon typ x2 (Storage st' (Address env)) | _ ⇒ False"
          using a10 a20 2(1) unfolding TypeSafe_def typeCompat_def by (cases t; force)
      qed
    qed
    have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof(intros)
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st) (Memory st') (Storage st' (Address env)) cd t l'"
      proof(cases l')
        case (Stackloc x1)
        then show ?thesis
        proof(cases "x1= l")
          case True
          then obtain t''' where tdef:"t = type.Memory t'''" using lDen a10 Stackloc 2(1) by blast
          then have x2IsP:"∃p. accessStore x1 (Stack st) = Some (KMemptr p)"
          proof(cases "accessStore x1 (Stack st)")
            case None
            then show ?thesis using 8 True 2(1) a10 Stackloc unfolding TypeSafe_def typeCompat_def by
force
          next
            case (Some a)
            then show ?thesis using tdef 8 True 2(1) a10 Stackloc unfolding TypeSafe_def
typeCompat_def
```

```
            by (cases a, force+ )
        qed
        then show ?thesis using a10 Stackloc lDen memorySame pMCon True tdef x2IsP 2(1) unfolding
TypeSafe_def typeCompat_def by force
      next
        case False
        then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')"
          using stackChanged by auto
        then show ?thesis using tcN a10 Stackloc unfolding typeCompat_def by force
      qed
    next
      case (Storeloc x2)
      then show ?thesis using tcN a10 unfolding typeCompat_def by force
    qed
  qed
next
  show "unique_locations (Denvalue env)" using 2(1) typeSafeUnique by auto
next
  have "(Accounts st) = Accounts(st')" using 8 by simp
  then show "balanceTypes (Accounts st')" using balanceTypes_def balanceTypes_def 2(1)
typeSafeAccounts by simp
next
  have a0:"compPointers (Stack st) (Denvalue env)" using 2(1) storageSame unfolding TypeSafe_def by
simp
  show " compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
  proof(intros)
    fix tp1 tp2 l1 l22 l1' l2' stl1 stl2
    assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
   (type.Storage tp2, l22) |∈| fmran (Denvalue env) ∧
   (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
   (l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l22 = Storeloc stl2)"
    have a2:"(∀x y. x |∈| fmran (Denvalue env) ∧ y |∈| fmran (Denvalue env) ∧ snd x = snd y ⟶ x =
y)"
      using 2(1) unfolding TypeSafe_def unique_locations_def by simp

    have a3:"l1 ≠ Stackloc l ∧ l22 ≠ Stackloc l" using a2 t'InDen a1 by auto

    then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
    proof(cases "l1 = Storeloc stl1")
      case t1:True
      then show ?thesis
      proof(cases "l22 = Storeloc stl2")
        case True
        then show ?thesis using t1 a0 a1 unfolding compPointers_def by blast
      next
        case False
        then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged a3
a1 by simp
        then show ?thesis using a0 a1 t1 unfolding compPointers_def by metis
      qed
    next
      case f1:False
      then have l1Same:"accessStore l1' (Stack st') = accessStore l1' (Stack st)"
        using stackChanged a3 a1 by simp
      then show ?thesis
      proof(cases "l22 = Storeloc stl2")
        case True
        then show ?thesis using f1 a0 a1 l1Same unfolding compPointers_def by metis
      next
        case False
        then have "accessStore l2' (Stack st') = accessStore l2' (Stack st)"
          using stackChanged a3 a1 by simp
        then show ?thesis using a0 a1 l1Same unfolding compPointers_def by metis
```

```
      qed
    qed

  qed
 next
  show "svalueTypes (Svalue env)" using svalueTypes_def typeSafeSvalue 2(1) by simp
 next
  have "(Storage st') = (Storage st)" using 8(4) by simp
  then show "safeContract (Accounts st') (Storage st')" using 2(1) 8 unfolding safeContract_def
TypeSafe_def by auto
 next
  have a10:"Toploc (Stack st') = Toploc (Stack st)" using 8(4) unfolding updateStore_def by simp
  then have a20:"∃ val. accessStore l (Stack st) = Some val" using t'InDen typeSafeLocExists 2(1)
TypeSafe_def by blast
  then have a30:"(∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st).
LSubPrefL2 loc (ShowL_nat tloc)))"
    using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  then have a40:"(∀ tloc loc. Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                 ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st').
LSubPrefL2 loc (ShowL_nat tloc)))"
    using a10 by simp
  show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
  proof intros
    fix tloc loc
    assume *:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
    then show "accessStore loc (Stack st') = None"
    proof(cases "loc = l")
      case True
      then show ?thesis using * a10
        by (metis a20 a30 option.distinct(1))
    next
      case False
      then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 8(4) unfolding
updateStore_def accessStore_def by simp
      then show ?thesis using 2(1) a40 * a10 False a30 by simp
    qed
  next
    fix loc y
    assume *:" accessStore loc (Stack st') = Some y "
    show "∃ tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)"
    proof(cases "loc = l")
      case True
      then show ?thesis using *a10 a20 a30 by simp
    next
      case False
      then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 8(4) unfolding
updateStore_def accessStore_def by simp
      then show ?thesis using * 2(1) a40 by simp
    qed
  qed
 next
  show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def using 8 by auto
 next
  show "lessThanTopLocs (Memory st')" using 2(1) unfolding TypeSafe_def using 8 by auto
 next
  have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by auto
  then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
 next
  show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using 8 by simp
 next
  have "Accounts st'= Accounts st" using 8 by auto
```

```
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 8 unfolding
fullyInitialised_def accessStore_def updateStore_def by auto
  next
    show "denvalueTypeCorrectness env (Stack st') (Memory st')" unfolding denvalueTypeCorrectness_def
    proof intros
      fix tt ll ptr_loc sub_loc
      assume *:"(type.Memory tt, Stackloc ll) |∈| fmran (Denvalue env) ∧ accessStore ll (Stack st') =
Some (KMemptr ptr_loc)"
      have SameMemory:"(Memory st) = (Memory st')" using 8 by simp
      have old: "denvalueTypeCorrectness env (Stack st) (Memory st)"
        using 2(1) unfolding TypeSafe_def by blast

      show "case tt of MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory
st') = Some arr)

     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val) "
      proof(cases "ll = l")
        case True
        then have isMTAr:"tt = (MTArray x t)" using lDen * by auto
        then have isP:"ptr_loc = p" using 8 * True by simp

        then have old2:"∀sub_loc. (case pParentT of
                        MTArray len arr ⇒
                          (∀i<len. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st) = Some
arr)

                        | MTValue val ⇒ accessTypeStore pParentPtr (Memory st) = Some (MTValue val))"
          using pOrigin old unfolding denvalueTypeCorrectness_def by auto
        have "(∀i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some t)"
        proof(cases "KMemptr p = KMemptr pParentPtr")
          case True
          then have "pParentT = MTArray x t" using pOrigin by auto
          then have "(∀i<x. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st) = Some t)"
            using old2 by simp
          then have "(∀i<x. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st') = Some t)"
            using SameMemory by (auto split:option.splits mtypes.splits)
          then show ?thesis using True isP SameMemory by blast
        next
          case False
          then obtain len arr where pt:"(extractValueType (KMemptr p) ≠ pParentPtr ∧ pParentT =
MTArray len arr ∧
       CompMemType (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))) len arr (MTArray
x t) pParentPtr
        (extractValueType (KMemptr p)))" using pOrigin by blast
          then have cmpt:" CompMemType (Memory st) len arr (MTArray x t) pParentPtr p" by simp

          have "((case pParentT of MTArray len arr ⇒
(∀i<len. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore pParentPtr (Memory st) = Some (MTValue val)))"
            using 2(1) pOrigin unfolding TypeSafe_def denvalueTypeCorrectness_def by auto
          then have pPrntAccT:"∀i<len. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st) =
Some arr"
            using pt by simp

          then have o2:"(∀locs tp.
      accessTypeStore locs (Memory st) = Some tp ⟶
      (case accessStore locs (Memory st) of None ⇒ False
       | Some (MValue v) ⇒ ∃val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs
(Memory st) = Some tp
       | Some (MPointer p) ⇒ ∃len arr. MCon tp (Memory st) p ∧ tp = MTArray len arr
                             ∧ (∀i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some arr)
                             ))"
            using 2(1) unfolding TypeSafe_def subPrefixStructuralConsistency_def using pt by blast
          have mcPrnt:"MCon (MTArray len arr) (Memory st) pParentPtr"
```

```
          using pt pOrigin 2(1) sameMemTSafe[OF 2(1), of pParentPtr] by auto

        have conc1:" ∀i<x. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some t"
          using CompMemTypeSubIndexes[OF cmpt _ mcPrnt pPrntAccT] 2(1) unfolding TypeSafe_def by simp
        then show ?thesis
          by (simp add: isP memorySame)
      qed

      then show ?thesis using isMTAr
        using mtypes.simps(5) <∀i<x. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some
t> by blast
    next
      case False
      then have "accessStore ll (Stack st) = accessStore ll (Stack st')" using stackChanged by blast
      then have "(case tt of
                  MTArray len arr ⇒ (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st)
= Some arr)

                  | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using old unfolding denvalueTypeCorrectness_def using * by auto
      then show ?thesis using SameMemory by metis
    qed
  qed
next
  show "subPrefixStructuralConsistency (Memory st')" using 8 2(1) unfolding TypeSafe_def by simp
next
  show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 8 by simp
next
  show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
    using 2(1) unfolding TypeSafe_def using 8 by simp
next
  show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MPointer v')"
    using memorySame by fastforce
next
  show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
    using memorySame by fastforce
next
  show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
      (Memory st) = None ⟹ accessStore loc (Memory st') = None"
    using memorySame by simp
next
  show "Toploc (Memory st) ≤ Toploc (Memory st')" using memorySame by auto
qed
qed
```

Assignment case 9: a memory array assigned to a storage variable reached through a stack pointer: the array is copied into storage.

```
lemma TypeSafe_Assign_9:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KMemptr p, type.Memory (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStackloc l, type.Storage t'),g')"
    and "cps2mTypeCompatible t' (MTArray x t)"
    and "accessStore l (Stack st) = Some (KStoptr p')"
    and "cpm2s p p' x t (Memory st) (Storage st (Address env)) = Some s"
    and "st' = st(|state.Gas := g', Storage := (Storage st) (Address env := s)|)"
  shows "StateInvariant env st st' cd"
proof -
```

```
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 9 = assms(4-9)

  obtain t''' where t''Def:"t' = STArray x t''' " using 9(3)
    by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
  have sameStack:"(Stack st) = (Stack st')" using 9 unfolding accessStore_def updateStore_def by auto
  have sameMemory:"Memory st' = Memory st " using 9 by auto
  have lInDen:"(type.Storage t', Stackloc l) |∈| fmran (Denvalue env)" using lexpStackloc_imps_inDen[of
lv env cd ] 9(2) by simp
  have nonLocChanged:"∀ t' locs. ¬ LSubPrefL2 locs p' ∨ locs = p' ⟶ accessStorage t' locs (Storage st
(Address env)) = accessStorage t' locs s"
    using 9 unfolding cpm2s_def using cpm2sSingleChange[of p p' t "Memory st" "(Storage st (Address
env))" x s]
    by fastforce
  have a30:" ∀ locs t' t''.
     cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray
x t') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs s"
    using 9(5) unfolding cpm2s_def using cpm2sSingleChange_typedPref[of p "p'" t "Memory st" "(Storage
st (Address env))" x s ] by simp
  then have a35:"∀ locs t''. locs ≠ p' ∧ ¬ TypedStoSubpref locs p' (STArray x t''') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs (Storage st' (Address
env))"
    using 9 t''Def by auto
  have mInStd:"s = Storage st' (Address env)" using 9 by simp

  have p'MCon:"SCon t' p' (Storage st (Address env))" using 2 9 unfolding TypeSafe_def typeCompat_def
using lInDen
    by (auto split:denvalue.splits option.splits stackvalue.splits type.splits)

  have MConsrc:"MCon (MTArray x t) (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)))
(extractValueType (KMemptr p)) ∧
      (∃ xx. KMemptr p = KMemptr xx) ∧
      (∃ stloc tp'' pa.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
          accessStore stloc (Stack (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))) = Some
(KMemptr pa) ∧
          (tp'' = (MTArray x t) ∧ KMemptr p = KMemptr pa ∨
           (∃ len arr.
               extractValueType (KMemptr p) ≠ pa ∧
               tp'' = MTArray len arr ∧ CompMemType (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st|))) len arr (MTArray x t) pa (extractValueType (KMemptr p)))))"
    using 2(1) 9(1) exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st|))"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KMemptr p" "type.Memory (MTArray x t)" g]
      2(3) unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀ loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have SCondest:"SCon (STArray x t''') p' s" using 9 unfolding cpm2s_def
    using MCon_cpm2s[of p p' t "Memory st" "(Storage st (Address env))" x s t'''] t''Def MConsrc
extractValueType.simps(2) by simp
  then have SCondest2:"SCon (STArray x t''') p' (Storage st' (Address env))"
    by (simp add: mInStd)
  then have SCondest3:"SCon t' p' (Storage st' (Address env))"
    by (simp add: mInStd t''Def)

  have stackDenvalLimits:"∀ struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
```

```
env)
                          ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto

  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 9 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
     (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2
= Stackloc l2'
         ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then show " if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "TypedStoSubpref stl2 stl1 tp1")
        case True
        then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref stl1 stl2 tp2")
          case True
          then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
        next
          case False
          then show ?thesis using f1 by simp
        qed
      qed
    qed
  next
    have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp

    show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
    proof intros
      fix e ct dud i tp
      assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
               ep $$ Contract (e::environment) = Some (ct, dud) ∧
               ct $$ i = Some (Var tp)"
      show "SCon tp i (Storage st' (Address e))"
      proof (cases "Address e = Address env")
        case False
        then have sameAddr:"Storage st' (Address e) = Storage st (Address e)" using 9 by simp
        moreover have "SCon tp i (Storage st (Address e))"
          using * scOld 9 unfolding safeContract_def by fastforce
        ultimately show ?thesis by simp
      next
        case addrEq:True
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * 9 by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
          using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
        have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
```

```
          using denI by (simp add: fmranI)
        have oldSConE:"SCon tp i (Storage st (Address e))"
          using scOld typedOld epOld ctOld unfolding safeContract_def by blast
        have oldSCon:"SCon tp i (Storage st (Address env))"
          using oldSConE addrEq by simp
        have cmpPtr:"(if TypedStoSubpref p' i tp then CompStoType tp t' i p'
                      else if TypedStoSubpref i p' t' then CompStoType t' tp p' i else True)"
          using 2(1) 9(3) inDenI lInDen t''Def unfolding TypeSafe_def compPointers_def
          using "9"(4) by blast
        have newSCon:"SCon tp i (Storage st' (Address env))"
          using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
        show ?thesis using newSCon addrEq by simp
      qed
    qed
  next
    show "balanceTypes (Accounts st')" using 9 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Stack st) = None) ∧
           (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 9 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" using sameStack * ** unfolding lessThanTopLocs_def by auto
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    show "lessThanTopLocs (Memory st')" using sameMemory unfolding lessThanTopLocs_def
      by (simp add: limitSt limitSt1)
  next
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"

      show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using sameStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(17)
)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(18))
        next
          case (KMemptr x3)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack sameMemory unfolding
TypeSafe_def typeCompat_def by (cases t''; fastforce)
        next
```

```
          case (KStoptr x4)
          then obtain struct where structDef: "t'' = type.Storage struct"
            using Stackloc adef inDen Stackloc 2(1) sameStack
            unfolding TypeSafe_def typeCompat_def
            by (cases t''; fastforce)

          have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t'''), Stackloc l) |∈| fmran (Denvalue env) ∧
            (l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4) ∨ l' = Storeloc x4) ∧
            (accessStore l (Stack st) = Some (KStoptr p')) ⟶
            (if TypedStoSubpref p' x4 struct then CompStoType struct (STArray x t''') x4 p'
             else if TypedStoSubpref x4 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x4 else True))"
            using 2(1) 9(3) lInDen inDen Stackloc structDef KStoptr unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref p' x4 struct then CompStoType struct (STArray x
t''') x4 p'
           else if TypedStoSubpref x4 p' (STArray x t''') then CompStoType (STArray x t''') struct p' x4
else True)"
            using 9(4) lInDen inDen Stackloc structDef KStoptr t''Def
            by (simp add: adef sameStack)

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Stackloc adef inDen Stackloc 2(1) sameStack structDef KStoptr
            by fastforce

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref p' x4 struct")
            case True
            then have "CompStoType struct (STArray x t''') x4 p'" using cmpStoPtr2 by simp
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast

          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x4 p' (STArray x t''')")
              case True
              then have "CompStoType t' struct p' x4 " using cmpStoPtr2 f1 t''Def by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs by blast

            next
              case False
              have notSame:"x4 ≠ p'" using False by auto
              have k7:"∀ locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ p' " using f1 by blast
              have k8:"∀ locs. TypedStoSubpref locs p' (STArray x t''') ⟶ locs ≠ x4" using False by
auto
              show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
            qed
          qed
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack KStoptr structDef by
simp
        qed

      next
        case (Storeloc x2)
        then obtain struct where structDef: "t'' = type.Storage struct" using Storeloc inDen 2(1)
sameStack unfolding TypeSafe_def typeCompat_def
          by (cases t''; fastforce)

        have cmpStoPtr:"(
```

```
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t'''), Stackloc l) |∈| fmran (Denvalue env) ∧
             l' = Storeloc x2) ∧
            (accessStore l (Stack st) = Some (KStoptr p')) ⟶
            (if TypedStoSubpref p' x2 struct then CompStoType struct (STArray x t''') x2 p'
             else if TypedStoSubpref x2 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x2 else True)"
          using 2(1) 9(3) lInDen inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by
blast

        then have cmpStoPtr2:"(if TypedStoSubpref p' x2 struct then CompStoType struct (STArray x t''')
x2 p'
             else if TypedStoSubpref x2 p' (STArray x t''') then CompStoType (STArray x t''') struct p'
x2 else True)"
          using 9(4) lInDen inDen Storeloc structDef t''Def by (simp add: sameStack)

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases "TypedStoSubpref p' x2 struct")
          case True
          then have "CompStoType struct (STArray x t''') x2 p'" using cmpStoPtr2 by simp
          then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref x2 p' (STArray x t''')")
            case True
            then have "CompStoType t' struct p' x2 " using cmpStoPtr2 f1 t''Def by simp
            then show ?thesis using SCondest3 SCon_imps_sublocs by blast
          next
            case False
            have notSame:"x2 ≠ p'" using False by auto
            have k7:"∀locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ p' " using f1 by blast
            have k8:"∀locs. TypedStoSubpref locs p' (STArray x t''') ⟶ locs ≠ x2" using False by
auto
            show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
          qed
        qed
        then show ?thesis using Storeloc inDen 2(1) sameStack Storeloc structDef by simp
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd" using
sameStack by simp
  next
    have "Accounts st'= Accounts st" using 9 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 9 unfolding
fullyInitialised_def by auto
  next
    have cc0:"∀l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
      using 9(6) unfolding updateStore_def accessStore_def by auto
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"

      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc sub_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
```

```
      MTArray len arr ⇒
         (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      moreover have "Memory st = Memory st'" using 9(6) by simp
      ultimately show "case t of
     MTArray len arr ⇒
      (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        by metis
    qed
  next
    show "subPrefixStructuralConsistency (Memory st')" using 2(1) unfolding TypeSafe_def using 9(6)
by auto
  next
    show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 9(6) by auto
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t "
      using 2(1) unfolding TypeSafe_def using 9(6) by auto
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 9(6) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 9(6) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
       (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 9(6) by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 9(6) by simp
  qed
qed
```

Assignment case 10: a memory array assigned to a storage location: the array is copied into storage.

```
lemma TypeSafe_Assign_10:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KMemptr p, type.Memory (MTArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStoreloc l, type.Storage t'),g')"
    and "cps2mTypeCompatible t' (MTArray x t)"
    and "cpm2s p l x t (Memory st) (Storage st (Address env)) = Some s"
    and "st' = st(|state.Gas := g', Storage := (Storage st) (Address env := s)|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 10 = assms(4-8)

  obtain t''' where t''Def:"t' = STArray x t''' " using 10(3)
    by (metis stypes.exhaust cps2mTypeCompatible.simps(2,4,6))
  have sameStack:"(Stack st) = (Stack st')" using 10 unfolding accessStore_def updateStore_def by auto
  have sameMemory:"Memory st' = Memory st " using 10 by auto
  have nonLocChanged:"∀t' locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStorage t' locs (Storage st
(Address env)) = accessStorage t' locs s"
    using 10(4) unfolding cpm2s_def using cpm2sSingleChange[of p l t "Memory st" "(Storage st (Address
env))" x s]
    by fastforce
```

```
  have a30:" ∀locs t' t''.
     cps2mTypeCompatible (STArray x t') (MTArray x t) ∧ locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x
t') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs s"
    using 10(4) unfolding cpm2s_def using cpm2sSingleChange_typedPref[of p "l" t "Memory st" "(Storage
st (Address env))" x s ]
    by simp
  then have a35:"∀locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t''') ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs (Storage st' (Address
env))"
    using 10 t''Def by auto
  have mInStd:"s = Storage st' (Address env)" using 10 by simp

  have MConsrc:"MCon (MTArray x t) (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)))
(extractValueType (KMemptr p)) ∧
      (∃xx. KMemptr p = KMemptr xx) ∧
      (∃stloc tp'' pa.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
          accessStore stloc (Stack (st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))) = Some
(KMemptr pa) ∧
          (tp'' = (MTArray x t) ∧ KMemptr p = KMemptr pa ∨
           (∃len arr.
               extractValueType (KMemptr p) ≠ pa ∧
               tp'' = MTArray len arr ∧ CompMemType (Memory (st(|Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st|))) len arr (MTArray x t) pa (extractValueType (KMemptr p)))))"
    using 2(1) 10(1) exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st|))"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KMemptr p" "type.Memory (MTArray x t)" g]
    using 2(3) unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have SCondest:"SCon (STArray x t''') l s" using 10 unfolding cpm2s_def
    using MCon_cpm2s[of p l t "Memory st" "(Storage st (Address env))" x s t'''] t''Def MConsrc
extractValueType.simps(2) by auto
  then have SCondest2:"SCon (STArray x t''') l (Storage st' (Address env))"
    by (simp add: mInStd)
  then have SCondest3:"SCon t' l (Storage st' (Address env))"
    by (simp add: mInStd t''Def)

  have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                        ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto

  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 10 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
     (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2
```

```
= Stackloc l2'
        ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then show " if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "TypedStoSubpref stl2 stl1 tp1")
        case True
        then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref stl1 stl2 tp2")
          case True
          then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
        next
          case False
          then show ?thesis using f1 by simp
        qed
      qed
    qed
  next
    have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp

    show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
    proof intros
      fix e ct dud i tp
      assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
               ep $$ Contract (e::environment) = Some (ct, dud) ∧
               ct $$ i = Some (Var tp)"
      show "SCon tp i (Storage st' (Address e))"
      proof (cases "Address e = Address env")
        case False
        then have sameAddr:"Storage st' (Address e) = Storage st (Address e)" using 10 by simp
        moreover have "SCon tp i (Storage st (Address e))"
          using * scOld 10 unfolding safeContract_def by fastforce
        ultimately show ?thesis by simp
      next
        case addrEq:True
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * 10 by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
          using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
        have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
          using denI by (simp add: fmranI)
        have oldSConE:"SCon tp i (Storage st (Address e))"
          using scOld typedOld epOld ctOld unfolding safeContract_def by blast
        have oldSCon:"SCon tp i (Storage st (Address env))"
          using oldSConE addrEq by simp
        have uniq:"unique_locations (Denvalue env)"
          using 2(1) typeSafeUnique by simp
        have epEnv:"ep $$ Contract env = Some (ct, dud)"
          using epOld addrEq
          using 2(3) fullyInitialised_def typedOld by auto
        show ?thesis
        proof(cases rule:lexpStorageG[OF 2(1) 10(2) 2(3)])
          case lInDen:1
          have inDenL:"(type.Storage t', Storeloc l) |∈| fmran (Denvalue env)"
            using lInDen by simp
          have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                       else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
            using 2(1) 10(3) inDenI inDenL unfolding TypeSafe_def compPointers_def by blast
```

```
  have newSCon:"SCon tp i (Storage st' (Address env))"
    using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
  show ?thesis using newSCon addrEq by simp
next
  case sub2:(2 l''' t)
  have inDenL:"(type.Storage t, Storeloc l''') |∈| fmran (Denvalue env)" using sub2 by simp
  have relL:"TypedStoSubpref l l''' t ∧ CompStoType t t' l''' l" using sub2 by simp
  show ?thesis
  proof(cases "i = l'''")
    case True
    have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage t, Storeloc l''')"
      using uniqueLocs[OF uniq inDenI inDenL] True by simp
    then have tpEq:"tp = t" by simp
    have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                 else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
      using relL tpEq True by simp
    have newSCon:"SCon tp i (Storage st' (Address env))"
      using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
    show ?thesis using newSCon addrEq by simp
  next
    case False
    have ctL:"ct $$ l''' = Some (Var t)"
      using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenL] .
    have nPrnt1:"¬TypedStoSubpref i l''' t"
        and nPrnt2:"¬TypedStoSubpref l''' i tp"
      using methodVarsNoPref False epOld ctOld ctL by blast+
    have n1:"¬TypedStoSubpref l i tp"
      using NotRelatedPrnt_imps_notRelatedChild[OF nPrnt1 nPrnt2] relL by blast
    have n2:"¬TypedStoSubpref i l t'"
      using NotReachablePrnt_imps_notReachableChild nPrnt1 nPrnt2 relL by blast
    have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                 else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
      using n1 n2 by simp
    have newSCon:"SCon tp i (Storage st' (Address env))"
      using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
    show ?thesis using newSCon addrEq by simp
  qed
next
  case sub3:(3 l' t l'')
  have inDenStk:"(type.Storage t, Stackloc l'') |∈| fmran (Denvalue env)" using sub3 by simp
  have ptrL:"accessStore l'' (Stack st) = Some (KStoptr l')" using sub3 by simp
  have rel0:"TypedStoSubpref l l' t ∧ CompStoType t t' l' l" using sub3 by simp
  obtain tprnt lprnt where
      inDenPr:"(type.Storage tprnt, Storeloc lprnt) |∈| fmran (Denvalue env)"
      and compPr:"CompStoType tprnt t lprnt l'"
    using fiPtr_parent_from_fullyInitialised[OF 2(3) inDenStk ptrL] by blast
  have compPr2:"CompStoType tprnt t' lprnt l"
    using compPr rel0 CompStoType_trns by blast
  have relPr:"TypedStoSubpref l lprnt tprnt ∧ CompStoType tprnt t' lprnt l"
    using compPr2 CompStoType_imps_TypedStoSubpref by blast
  show ?thesis
  proof(cases "i = lprnt")
    case True
    have pairEq:"(type.Storage tp, Storeloc i) = (type.Storage tprnt, Storeloc lprnt)"
      using uniqueLocs[OF uniq inDenI inDenPr] True by simp
    then have tpEq:"tp = tprnt" by simp
    have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                 else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
      using compPr2 tpEq True
      using relPr by presburger
    have newSCon:"SCon tp i (Storage st' (Address env))"
      using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
    show ?thesis using newSCon addrEq by simp
  next
```

```
            case False
            have ctPr:"ct $$ lprnt = Some (Var tprnt)"
              using fi_denvalue_storeloc_to_contract_var[OF 2(3) epEnv inDenPr] .
            have nPrnt1:"¬TypedStoSubpref i lprnt tprnt"
                and nPrnt2:"¬TypedStoSubpref lprnt i tp"
              using methodVarsNoPref False epOld ctOld ctPr by blast+
            have n1:"¬TypedStoSubpref l i tp"
              using NotRelatedPrnt_imps_notRelatedChild[OF nPrnt1 nPrnt2] relPr by blast
            have n2:"¬TypedStoSubpref i l t'"
              using NotReachablePrnt_imps_notReachableChild nPrnt1 nPrnt2 relPr by blast
            have cmpPtr:"(if TypedStoSubpref l i tp then CompStoType tp t' i l
                          else if TypedStoSubpref i l t' then CompStoType t' tp l i else True)"
              using n1 n2 by simp
            have newSCon:"SCon tp i (Storage st' (Address env))"
              using SCon_update_array_subloc_cases[OF cmpPtr SCondest2 SCondest3 oldSCon t''Def a35] .
            show ?thesis using newSCon addrEq by simp
          qed
        qed
      qed
    qed
  next
    show "balanceTypes (Accounts st')" using 10 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Stack st) = None) ∧
           (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 10 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" using sameStack * ** unfolding lessThanTopLocs_def by auto
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    show "lessThanTopLocs (Memory st')" using sameMemory unfolding lessThanTopLocs_def
      by (simp add: limitSt limitSt1)
  next
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"

      show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using sameStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(17)
)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
```

```
stackvalue.simps(18))
        next
          case (KMemptr x3)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack sameMemory unfolding
TypeSafe_def typeCompat_def by (cases t''; fastforce)
        next
          case (KStoptr x4)
          then obtain struct where structDef: "t'' = type.Storage struct" using Stackloc adef inDen
Stackloc 2(1) sameStack unfolding TypeSafe_def typeCompat_def
            by (cases t''; fastforce)

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases rule:lexpStorageG[OF 2(1) 10(2) 2(3)])
            case lInDen:1
            have cmpStoPtr:"(
              (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
              (type.Storage (STArray x t'''), Storeloc l) |∈| fmran (Denvalue env) ∧
               l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
               ⟶
              (if TypedStoSubpref l x4 struct then CompStoType struct (STArray x t''') x4 l
               else if TypedStoSubpref x4 l (STArray x t''') then CompStoType (STArray x t''') struct l
x4 else True)"
              using 2(1) 10(3) lInDen inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

            then have cmpStoPtr2:"(if TypedStoSubpref l x4 struct then CompStoType struct (STArray x
t''') x4 l
               else if TypedStoSubpref x4 l (STArray x t''') then CompStoType (STArray x t''') struct l
x4 else True)"
              using 10(3) lInDen inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

            have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
              by fastforce

            have "SCon struct x4 (Storage st' (Address env))"
            proof(cases "TypedStoSubpref l x4 struct")
              case True
              then have "CompStoType struct (STArray x t''') x4 l" using cmpStoPtr2 by simp
              then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
                by blast

            next
              case f1:False
              then show ?thesis
              proof(cases "TypedStoSubpref x4 l (STArray x t''')")
                case True
                then have "CompStoType t' struct l x4 " using cmpStoPtr2 f1 t''Def by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs by blast
              next
                case False
                have notSame:"x4 ≠ l" using False by auto
                have k7:"∀locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 by blast
                have k8:"∀locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x4" using False by
auto
                show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
              qed
            qed
            then show ?thesis by simp
          next
            case sub2:(2 l''' t)
            have cmpStoPtr:"(
              (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
```

```
            (type.Storage t, Storeloc l''') |∈| fmran (Denvalue env) ∧
             l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
             ⟶
            (if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
           using 2(1) 10(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

         then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
            else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
           using 10(3) sub2 inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

         have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
           by fastforce
         have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

         have "SCon struct x4 (Storage st' (Address env))"
         proof(cases "TypedStoSubpref l''' x4 struct")
           case True
           then have "CompStoType struct t x4 l'''" using cmpStoPtr2 by simp
           then have "CompStoType struct (STArray x t''') x4 l" using sub2
             using CompStoType_trns t''Def by blast
           then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
             by blast
         next
           case f1:False
           then show ?thesis
           proof(cases "TypedStoSubpref x4 l''' t")
             case True
             then have "l''' ≠ x4" using f1
               using TypedStoSubpref_sameLoc by auto
             then show ?thesis
             proof(cases "l = x4")
               case t5:True
               then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
               moreover have "CompStoType t t' l''' x4" using t5 sub2 by auto
               ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                 by simp
               then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
             next
               case False
               then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
               then show ?thesis
               proof(cases "CompStoType t' struct l x4 ")
                 case True
                 then show ?thesis
                   using SCon_imps_sublocs SCondest3 by blast
               next
                 case False
                 then show ?thesis
                   using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t struct l'''
x4› a35 scl''' sub2(2) t''Def by blast
                 qed
               qed
           next
             case False
             then have asm10:"¬ TypedStoSubpref l x4 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub2 by blast
             then have asm20:"¬ TypedStoSubpref x4 l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
             have notSame:"x4 ≠ l" using False sub2 by blast
```

```
              have k7:"∀locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 sub2 asm10
asm20 by blast
              have k8:"∀locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x4" using False
sub2 t''Def asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
            qed
          qed

          then show ?thesis by simp
        next
          case sub3:(3 l''' t l'''')
          have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage t, Stackloc l'''') |∈| fmran (Denvalue env) ∧
             l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4)) ∧ accessStore l''''
(Stack st) = Some (KStoptr l''')
              ⟶
            (if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
            using 2(1) 10(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t x4 l'''
             else if TypedStoSubpref x4 l''' t then CompStoType t struct l''' x4 else True)"
            using 10(3) sub3 inDen adef Stackloc KStoptr structDef t''Def sameStack by auto

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
            by fastforce
          have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x4 struct")
            case True
            then have "CompStoType struct t x4 l'''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t''') x4 l" using sub3
              using CompStoType_trns t''Def by blast
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x4 l''' t")
              case True
              then have "l''' ≠ x4" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x4")
                case t5:True
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                moreover have "CompStoType t t' l''' x4" using t5 sub3 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
                then have " CompStoType t struct l''' x4" using cmpStoPtr2 f1 t''Def True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x4 ")
                  case True
                  then show ?thesis
```

```
                    using SCon_imps_sublocs SCondest3 by blast
                  next
                    case False
                    then show ?thesis
                      using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t struct l'''
x4> a35 scl''' sub3(3) t''Def by blast
                  qed
                qed
              next
                case False
                then have asm10:"¬ TypedStoSubpref l x4 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
                then have asm20:"¬ TypedStoSubpref x4 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
                have notSame:"x4 ≠ l" using False sub3 by blast
                have k7:"∀locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 sub3 asm10
asm20 by blast
                have k8:"∀locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x4" using False
sub3 t''Def asm10 asm20 by blast
                show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
              qed
            qed
            then show ?thesis by simp
          qed
          then show ?thesis using Stackloc inDen 2(1) sameStack KStoptr adef structDef by simp
        qed
      next
        case (Storeloc x2)
        then obtain struct where structDef: "t'' = type.Storage struct" using Storeloc inDen 2(1)
sameStack unfolding TypeSafe_def typeCompat_def
          by (cases t''; fastforce)

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases rule:lexpStorageG[OF 2(1) 10(2) 2(3)])
          case lInDen:1
          have cmpStoPtr:"(
            (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t'''), Storeloc l) |∈| fmran (Denvalue env) ∧
             l' = Storeloc x2)
             ⟶
            (if TypedStoSubpref l x2 struct then CompStoType struct (STArray x t''') x2 l
             else if TypedStoSubpref x2 l (STArray x t''') then CompStoType (STArray x t''') struct l x2
else True)"
            using 2(1) 10(3) lInDen inDen Storeloc structDef unfolding TypeSafe_def compPointers_def
by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l x2 struct then CompStoType struct (STArray x
t''') x2 l
             else if TypedStoSubpref x2 l (STArray x t''') then CompStoType (STArray x t''') struct l x2
else True)"
            using 10(3) lInDen inDen Storeloc structDef t''Def sameStack by simp

          have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
            by fastforce

          have "SCon struct x2 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l x2 struct")
            case True
            then have "CompStoType struct (STArray x t''') x2 l" using cmpStoPtr2 by simp
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
```

```
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x2 l (STArray x t''')")
              case True
              then have "CompStoType t' struct l x2 " using cmpStoPtr2 f1 t''Def by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs by blast
            next
              case False
              have notSame:"x2 ≠ l" using False by auto
              have k7:"∀locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 by blast
              have k8:"∀locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x2" using False by
auto
              show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
            qed
          qed
          then show ?thesis by simp
        next
          case sub2:(2 l''' t)
          have cmpStoPtr:"(
              (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
              (type.Storage t, Storeloc l''') |∈| fmran (Denvalue env) )
               ⟶
              (if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
               else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
            using 2(1) 10(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
               else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
            using 10(3) sub2 inDen Storeloc structDef t''Def sameStack by auto

          have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
            by fastforce
          have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

          have "SCon struct x2 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x2 struct")
            case True
            then have "CompStoType struct t x2 l''''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t''') x2 l" using sub2
              using CompStoType_trns t''Def by blast
            then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x2 l''' t")
              case True
              then have "l''' ≠ x2" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x2")
                case t5:True
                then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
                moreover have "CompStoType t t' l''' x2" using t5 sub2 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
```

```
              then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
              then show ?thesis
              proof(cases "CompStoType t' struct l x2 ")
                case True
                then show ?thesis
                  using SCon_imps_sublocs SCondest3 by blast
              next
                case False
                then show ?thesis
                  using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t struct l'''
x2> a35 scl''' sub2(2) t''Def by blast
              qed
            qed
          next
            case False
            then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub2 by blast
            then have asm20:"¬ TypedStoSubpref x2 l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
            have notSame:"x2 ≠ l" using False sub2 by blast
            have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub2 asm10 asm20
by blast
            have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x2" using False sub2
t''Def asm10 asm20 by blast
            show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
          qed
        qed

        then show ?thesis by simp
      next
        case sub3:(3 l''' t l'''')
        have cmpStoPtr:"(
            (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
            (type.Storage t, Stackloc l'''') |∈| fmran (Denvalue env) ∧
             accessStore l'''' (Stack st) = Some (KStoptr l'''))
             ⟶
            (if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
             else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
          using 2(1) 10(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t x2 l'''
             else if TypedStoSubpref x2 l''' t then CompStoType t struct l''' x2 else True)"
          using 10(3) sub3 inDen Storeloc structDef t''Def sameStack by auto

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce
        have scl''':"SCon t l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases "TypedStoSubpref l''' x2 struct")
          case True
          then have "CompStoType struct t x2 l''''" using cmpStoPtr2 by simp
          then have "CompStoType struct (STArray x t''') x2 l" using sub3
            using CompStoType_trns t''Def by blast
          then show ?thesis using SCondest2 SConx4Old a35 t''Def TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref x2 l''' t")
```

```
              case True
              then have "l''' ≠ x2" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x2")
                case t5:True
                then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
                moreover have "CompStoType t t' l''' x2" using t5 sub3 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
                then have " CompStoType t struct l''' x2" using cmpStoPtr2 f1 t''Def True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x2 ")
                  case True
                  then show ?thesis
                    using SCon_imps_sublocs SCondest3 by blast
                next
                  case False
                  then show ?thesis
                    using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t struct l'''
x2> a35 scl''' sub3(3) t''Def by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
              then have asm20:"¬ TypedStoSubpref x2 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x2 ≠ l" using False sub3 by blast
              have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub3 asm10 asm20
by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t''') ⟶ locs ≠ x2" using False sub3
t''Def asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 t''Def by
blast
            qed
          qed
          then show ?thesis by simp
        qed
        then show ?thesis using Storeloc inDen 2(1) sameStack Storeloc structDef by simp
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd " us-
ing sameStack by simp
  next
    have "Accounts st'= Accounts st" using 10 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 10 unfolding
fullyInitialised_def by auto
  next
    have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
      using 10(5) unfolding updateStore_def accessStore_def by auto
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"

      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc sub_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
```

```
      MTArray len arr ⇒
         (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      moreover have "Memory st = Memory st'" using 10(5) by simp
      ultimately show "case t of
     MTArray len arr ⇒
      (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        by metis
    qed
  next
    show "subPrefixStructuralConsistency (Memory st')" using 2(1) unfolding TypeSafe_def using 10(5)
by auto
  next
    show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 10(5) by auto
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t "
      using 2(1) unfolding TypeSafe_def using 10(5) by auto
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 10(5) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 10(5) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
       (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 10(5) by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 10(5)by simp
  qed
qed
```

Assignment case 11: a memory array assigned to a memory location: the location is updated to point to the same array.

```
lemma TypeSafe_Assign_11:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st⦇state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KMemptr p, type.Memory (MTArray x t)), g)"
    and "lexp lv env cd (st⦇state.Gas := g⦈) g = Normal((LMemloc l, type.Memory t'),g')"
    and "t' = MTArray x t"
    and "st' = st⦇state.Gas := g', Memory := updateStore l (MPointer p) (Memory st)⦈"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 11 = assms(4-7)
  have temp:"TypeSafe env (Accounts (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))
(Storage (st⦇Gas := g⦈)) cd"
    using 2(1) by simp
  have ttt:"fullyInitialised env (Accounts (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈))" using 2(3) unfold-
ing fullyInitialised_def by simp
  then show ?thesis
  proof(cases rule:lexpIndexMem[OF 11(2) temp ttt])
    case lInfo:(1 x21 x22 tp tParent l' l'' prnt len' arr' i)
    have sameStack:"(Stack st') = Stack st" using 11 unfolding accessStore_def updateStore_def by auto
    have sameStorage:"Storage st' = Storage st " using 11 by auto
```

```
    have accL:"accessStore l (Memory st') = Some (MPointer p)" using 11(4) by auto
    have nonLocChanged:"∀ locs. locs ≠ l ⟶ accessStore locs (Memory st') = accessStore locs (Memory
st)"
      using 11 unfolding updateStore_def accessStore_def by simp

    have MConsrc:"MCon (MTArray x t) (Memory (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd
st⦈)) (extractValueType (KMemptr p)) ∧
      (∃ xx. KMemptr p = KMemptr xx) ∧
      (∃ stloc tp'' pa.
          (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue env) ∧
          accessStore stloc (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some
(KMemptr pa) ∧
          (tp'' = (MTArray x t) ∧ KMemptr p = KMemptr pa ∨
           (∃ len arr.
               extractValueType (KMemptr p) ≠ pa ∧
               tp'' = MTArray len arr ∧ CompMemType (Memory (st⦇Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st⦈)) len arr (MTArray x t) pa (extractValueType (KMemptr p)))))"
      using 2(1) 11(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st⦈)"
          "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KMemptr p" "type.Memory (MTArray x t)" g]
      using 2(3) unfolding fullyInitialised_def
      by (auto split:type.splits if_splits )


    then obtain pParent pParentT pParentPtr where
      pOrigin:"MCon (MTArray x t) (Memory (st)) (extractValueType (KMemptr p)) ∧(type.Memory pParentT,
Stackloc pParent) |∈| fmran (Denvalue env) ∧
          accessStore pParent (Stack (st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)) = Some
(KMemptr pParentPtr) ∧
          (pParentT = (MTArray x t) ∧ KMemptr p = KMemptr pParentPtr ∨
           (∃ len arr.
               extractValueType (KMemptr p) ≠ pParentPtr ∧
               pParentT = MTArray len arr ∧ CompMemType (Memory (st⦇Gas := state.Gas st - costs (ASSIGN
lv ex) env cd
                   st⦈)) len arr (MTArray x t) pParentPtr (extractValueType (KMemptr p))))"
      by force
    then have l:"MCon pParentT (Memory st) pParentPtr" using sameMemTSafe[OF 2(1)] by auto

    have pTAccessOld:"(∀ i<x. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some t)"
    proof(cases "pParentT = (MTArray x t) ∧ KMemptr p = KMemptr pParentPtr")
      case True
      then show ?thesis using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def
        using pOrigin by fastforce
    next
      case False
      then obtain l1 a1 where l1Def:"(p ≠ pParentPtr ∧ pParentT = MTArray l1 a1
                ∧ CompMemType (Memory st) l1 a1 (MTArray x t) pParentPtr p)"
        using pOrigin by auto
      then have prntT:" ∀ i<l1. accessTypeStore (hash pParentPtr (ShowL_nat i)) (Memory st) = Some a1"
        using pOrigin using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def
        by fastforce
      have subPref:"subPrefixStructuralConsistency (Memory st)"
        using 2(1) unfolding TypeSafe_def by blast
      then show ?thesis using CompMemTypeSubIndexes[OF _ subPref _ prntT] l1Def
        using l by blast
    qed


    obtain len subT where tParentType:"tParent = MTArray len subT" using lInfo by blast
    then obtain p'' where lOrigin:"accessStore l (Memory (st)) = Some (MPointer p'')"
        and compType:"CompMemType (Memory (st⦇Gas := g⦈)) len subT (MTArray x t) l'' p''"
        and lsublocs:"l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = (MTArray x t) ∧ MCon (MTArray
len' arr') (Memory (st)) prnt"
        and lsublocs3:"(prnt = l'' ∧ len = len' ∧ arr' = subT ∨ CompMemType (Memory (st)) len subT
```

```
(MTArray len' arr') l'' prnt)"
      using lInfo 11(3) by force
    then have lsublocs2:" CompMemType (Memory (st)) len' arr' (MTArray x t) prnt p''"
      using 11(3) CompMemType.simps(2) by blast
    then have bb9:"∀subT subloc. CompMemType (Memory (st)) len' arr' subT prnt subloc ∧ subloc = p''
                                ⟶ subT = (MTArray x t)"
      using CompMemTypeSameLocsSameType lsublocs by blast

    have mconPrnt:"MCon (MTArray len' (MTArray x t)) (Memory st) prnt" using lsublocs by auto
    have ldef:"l = hash prnt (ShowL_nat i) ∧ i < len'" using lsublocs by auto

    have t6:"(type.Memory tParent, Stackloc l') |∈| fmran (Denvalue env)" using lInfo by blast
    have t7:" MCon (MTArray len subT) (Memory (st)) l''" using lInfo
      using tParentType by auto
    then have mconlOld:"MCon (MTArray x t) (Memory st) p''" using 11(3) lInfo lOrigin by auto

    have mcP:"MCon (MTArray x t) (Memory st) p" using MConsrc by force
    have compP:"¬CompMemType (Memory (st)) x t (MTArray len' (MTArray x t)) p prnt"
      using MCon_subTypes_imps_noPrnt mcP mconPrnt
      by simp
    have pNotPrnt:"p ≠ prnt" using mcP mconPrnt nonLocChanged
      using MConSubTypes CompMemJustType.simps(2) by blast
    have pSubs:"TypedMemSubPrefPtrs (Memory st) len' (MTArray x t) prnt p ⟶ (∃i'. accessStore (hash
prnt
       (ShowL_nat i')) (Memory st) = Some (MPointer p) ∧ i'<len')"
    proof intros
      assume in1:"TypedMemSubPrefPtrs (Memory st) len' (MTArray x t) prnt p"
      then obtain i2 l2 where i2Def: "i2<len' ∧ accessStore (hash prnt (ShowL_nat i2)) (Memory st) =
Some
         (MPointer l2) ∧ (l2 = p ∨ TypedMemSubPrefPtrs (Memory st) x t l2 p)"
        unfolding TypedMemSubPrefPtrs.simps by blast
      show "∃i'. accessStore (hash prnt (ShowL_nat i')) (Memory st) = Some (MPointer p) ∧ i' < len'"
      proof(cases "l2 = p")
        case True
        then show ?thesis using i2Def by auto
      next
        case False
        then have in2:"TypedMemSubPrefPtrs (Memory st) x t l2 p" using i2Def by simp
        have in3:"∀ta. CompMemJustType t ta ⟶ ¬ MCon ta (Memory st) p" using MConSubTypes[OF mcP]
by blast
        have "MCon (MTArray x t) (Memory st) l2" using i2Def mconPrnt
          using MCon_imps_sub_Mcon by blast

        then show ?thesis using in3 in2
        proof(induction t arbitrary:l2 x)
          case (MTArray x1 t)
          obtain i3 l3 where i3Def:"i3<x ∧ accessStore (hash l2 (ShowL_nat i3)) (Memory st) = Some
(MPointer l3)
                                  ∧ (l3 = p ∨ TypedMemSubPrefPtrs (Memory st) x1 t l3 p)"
            using MTArray.prems(3) unfolding TypedMemSubPrefPtrs.simps by auto
          then have mcL3:"MCon (MTArray x1 t) (Memory st) l3" using MTArray.prems(1)
            using MconSameTypeSameAccessWithTyping by blast
          then have notP:"l3 ≠ p" using MTArray.prems(3)
            using MConSubTypes MTArray.prems(2) CompMemJustType.simps(2) by blast
          have "∀taa. CompMemJustType t taa ⟶ ¬ MCon taa (Memory st) p" using MTArray.prems by simp
          moreover have "TypedMemSubPrefPtrs (Memory st) x1 t l3 p" using notP i3Def by simp
          ultimately show ?case using MTArray.IH mcL3 by blast
        next
          case (MTValue x')
          then show ?case
            by (metis CompTypeRemainsMCon TypedMemSubPrefPtrs.simps(1) CompMemJustType.simps(1)
CompMemType.simps(1))
        qed
      qed
```

```
    qed

    have p''ToPrnt:"¬TypedMemSubPrefPtrs (Memory st) x t p'' prnt"
      by (meson CompMemType_imps_CompMemJustType CompMemType_imps_TypedMemSubPrefPtrs
CompTypeRemainsMCon TypedMemSubPrefOneWay lsublocs lsublocs2)
    moreover have p''notprnt:"p'' ≠prnt" using lsublocs2
      using CompMemType_imps_TypedMemSubPrefPtrs TypedMemSubPrefPtrs_imps_notsame lsublocs by blast
    ultimately have mcP''New:"MCon (MTArray x t) (Memory st') p''" using mconlOld
limitedMemoryChange[of "Memory st" x t p'' prnt i ] nonLocChanged 11 ldef by fastforce

    have mcPrnt:"∃t''. MCon t'' (Memory st) prnt ∧ CompMemJustType t'' (MTArray x t) ∧ t'' ≠ (MTArray
x t)" using mconPrnt by fastforce
    have asf:"¬TypedMemSubPrefPtrs (Memory st) x t p prnt"
    proof
      assume in1:"TypedMemSubPrefPtrs (Memory st) x t p prnt"
      then show False using mcP mcPrnt
      proof(induction t arbitrary:x p)
        case (MTArray x1 t)
        then obtain i2 l2 where i2Def: "i2<x ∧ accessStore (hash p (ShowL_nat i2)) (Memory st) = Some
(MPointer l2)
                                        ∧ (l2 = prnt ∨ TypedMemSubPrefPtrs (Memory st) x1 t l2
prnt)"
          unfolding TypedMemSubPrefPtrs.simps by blast
        then have mcl2:"MCon (MTArray x1 t) (Memory st) l2" using MTArray.prems(2)
          using MCon_imps_sub_Mcon by blast
        then show ?case
        proof(cases "l2 = prnt")
          case True
          then have "MCon (MTArray x1 t) (Memory st) prnt" using mcl2 by simp
          then show ?thesis
          proof -
            obtain t'' where t'': "MCon t'' (Memory st) prnt" "CompMemJustType t'' (MTArray x (MTArray
x1 t))"
                "t'' ≠ MTArray x (MTArray x1 t)"
              using MTArray.prems(3) by blast
            then have "CompMemJustType t'' (MTArray x1 t)"
              using CompMemTypes_asc CompMemJustType_refl by fastforce
            then have "t'' = MTArray x1 t"
              using MCon_CompMemJustType_unique[OF CompMemJustType_refl _ t''(1)] ‹MCon (MTArray x1 t)
(Memory st) prnt› by blast
            then show False
              using t''(2,3) reversible_CompMemJustType_imps_same CompMemJustType_refl by fastforce
          qed
        next
          case False
          then have "TypedMemSubPrefPtrs (Memory st) x1 t l2 prnt" using i2Def by simp
          then show ?thesis using MTArray.IH[OF _ mcl2] MTArray.prems(3)
            by (metis CompMemTypes_asc reversible_CompMemJustType_imps_same CompMemJustType.simps(2))
        qed
      next
        case (MTValue x')
        then have "∃i<x. hash p (ShowL_nat i) = prnt" unfolding TypedMemSubPrefPtrs.simps by blast
        then show ?case using MTValue unfolding MCon.simps
          using lsublocs by fastforce
      qed
    qed

    have mcPNew:"MCon (MTArray x t) (Memory st') p" using mcP asf pNotPrnt
    proof(induction t arbitrary: x p)
      case (MTArray x1 t)
      then have xnotzero:"x ≠ 0" unfolding MCon.simps by simp
      have "∀i<x. (case accessStore (hash p (ShowL_nat i)) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False
                                | MTValue typ ⇒ MCon (MTArray x1 t) (Memory st') (hash p (ShowL_nat
```

```
i)))
            | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray x1 t)
(Memory st') loc2 | MTValue Types ⇒ False)) ∧
           ((∃p'. accessStore p (Memory st') = Some (MPointer p')) ∨ accessStore p (Memory st') =
None)"
      proof(intros)
        fix i2 assume i2Def:"i2<x"
        have "(hash p (ShowL_nat i2)) ≠ l"
        proof
          assume in1':"hash p (ShowL_nat i2) = l"
          then have "p = prnt" using ldef ShowLNatDot
            using hash_injective by blast
          then show False using MTArray.prems(3) by simp
        qed
        then have sameAccess:"accessStore (hash p (ShowL_nat i2)) (Memory st') = accessStore (hash p
(ShowL_nat i2)) (Memory st)" using nonLocChanged by simp
        then obtain p2 where p2Def: "accessStore (hash p (ShowL_nat i2)) (Memory st') = Some (MPointer
p2) ∧ MCon (MTArray x1 t) (Memory st) p2" using i2Def
          using MTArray.prems by (metis MConArrayPointers MCon_imps_sub_Mcon bot_nat_0.not_eq_extremum
xnotzero)
        then have "¬ TypedMemSubPrefPtrs (Memory st) x1 t p2 prnt" using MTArray.prems
          using i2Def sameAccess by fastforce
        moreover have "p2 ≠ prnt" using MTArray.prems p2Def i2Def
          by (simp add: sameAccess)
        ultimately have "MCon (MTArray x1 t) (Memory st') p2" using MTArray.IH[of x1 p2] p2Def
MTArray.prems by blast
        then show "case accessStore (hash p (ShowL_nat i2)) (Memory st') of None ⇒ False
     | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTArray
x1 t) (Memory st') (hash p (ShowL_nat i2)))
     | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray x1 t) (Memory
st') loc2 | MTValue Types ⇒ False)"
          using p2Def by simp
        show "(∃p'. accessStore p (Memory st') = Some (MPointer p')) ∨ accessStore p (Memory st') =
None"
        proof(cases "p = l")
          case True
          then show ?thesis using 11 by simp
        next
          case False
          then show ?thesis using nonLocChanged using MTArray.prems(1) unfolding MCon.simps
            using i2Def sameAccess by simp
        qed
      qed
      then show ?case using MCon.simps(2)[of x "(MTArray x1 t)" "Memory st'" p ] xnotzero
        by simp
    next
      case (MTValue x')
      then have xnotzero:"x ≠ 0" unfolding MCon.simps by simp
      have "∀i<x. (case accessStore (hash p (ShowL_nat i)) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒
                (case MTValue x' of MTArray l a ⇒ False
                | MTValue typ ⇒ (case accessStore (hash p (ShowL_nat i)) (Memory st') of None
                    ⇒ False | Some (MValue xa) ⇒ typeCon x' xa | Some (MPointer t) ⇒ False))
            | Some (MPointer loc2) ⇒
                (case MTValue x' of MTArray len' arr' ⇒ (case accessStore loc2 (Memory st') of None ⇒
False | Some
                    (MValue xa) ⇒ typeCon x' xa | Some (MPointer t) ⇒ False)
                | MTValue Types ⇒ False))∧
           ((∃p'. accessStore p (Memory st') = Some (MPointer p')) ∨ accessStore p (Memory st') =
None)"
      proof(intros)
        fix i2 assume i2Def:"i2<x"
        then have "(hash p (ShowL_nat i2)) ≠ l" using ldef
          by (metis MCon_sub_MTVal_imps_val MTValue.prems(1) memoryvalue.distinct(1) lOrigin
```

```
option.inject)
        then have sameAccess:" accessStore (hash p (ShowL_nat i2)) (Memory st') = accessStore (hash p
            (ShowL_nat i2)) (Memory st)" using nonLocChanged by simp
        then show "case accessStore (hash p (ShowL_nat i2)) (Memory st') of None ⇒ False
     | Some (MValue val) ⇒
         (case MTValue x' of MTArray l a ⇒ False
         | MTValue typ ⇒ (case accessStore (hash p (ShowL_nat i2)) (Memory st') of None ⇒
             False | Some (MValue xa) ⇒ typeCon x' xa | Some (MPointer t) ⇒ False))
     | Some (MPointer loc2) ⇒
         (case MTValue x' of MTArray len' arr' ⇒ (case accessStore loc2 (Memory st') of None ⇒ False |
Some
             (MValue xa) ⇒ typeCon x' xa | Some (MPointer t) ⇒ False)
         | MTValue Types ⇒ False)" using MTValue(1) i2Def mcon_accessStore xnotzero by (auto
split:option.splits memoryvalue.splits mtypes.splits)
        show "(∃p'. accessStore p (Memory st') = Some (MPointer p')) ∨ accessStore p (Memory st') =
None"
        proof(cases "p = l")
          case True
          then show ?thesis using 11 by simp
        next
          case False
          then show ?thesis using nonLocChanged using MTValue(1) unfolding MCon.simps
            using i2Def sameAccess by simp
        qed
      qed
      then show ?case unfolding MCon.simps using xnotzero
        by simp
    qed

    have mcPrntNew:"MCon (MTArray len' (MTArray x t)) (Memory st') prnt"
    proof -
      have lenNotZero:"len' ≠ 0" using lsublocs by auto
      have prntNotL:"prnt ≠ l" using ldef
        by (metis hash_inequality)
      then have p2:"(∃p. accessStore prnt (Memory st') = Some (MPointer p)) ∨ accessStore prnt (Memory
st') = None"
        using mconPrnt lenNotZero nonLocChanged by simp
      have "∀i<len'.
         (case accessStore (hash prnt (ShowL_nat i)) (Memory st') of None ⇒ False
          | Some (MValue val) ⇒ (case MTArray x t of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTArray
              x t) (Memory st') (hash prnt (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTArray x t of MTArray len' arr' ⇒ MCon (MTArray x t) (Memory
st') loc2 | MTValue Types ⇒ False))"
      proof(intros)
        fix i1 assume in1:"i1<len'"
        show "case accessStore (hash prnt (ShowL_nat i1)) (Memory st') of None ⇒ False
       | Some (MValue val) ⇒ (case MTArray x t of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTArray
x
           t) (Memory st') (hash prnt (ShowL_nat i1)))
       | Some (MPointer loc2) ⇒ (case MTArray x t of MTArray len' arr' ⇒ MCon (MTArray x t) (Memory
st') loc2 | MTValue Types ⇒ False)"
        proof(cases "i1 = i")
          case True
          then have "(hash prnt (ShowL_nat i1)) = l" using ldef by simp
          then have "accessStore (hash prnt (ShowL_nat i1)) (Memory st') = Some(MPointer p)" using 11
by simp
          then show ?thesis using mcPNew by simp
        next
          case False
          then have "accessStore (hash prnt (ShowL_nat i1)) (Memory st') = accessStore (hash prnt
(ShowL_nat i1)) (Memory st)" using ldef nonLocChanged in1
            by (metis Read_Show_nat'_id hash_never_equal_suffix)
          then obtain ptr where ptrDef:"accessStore (hash prnt (ShowL_nat i1)) (Memory st) = Some
```

```
(MPointer ptr)" using in1
          using MConArrayPointers lsublocs by blast
        then have ptrMC:"MCon (MTArray x t) (Memory st) ptr" using in1 mconPrnt
          using MCon_imps_sub_Mcon by blast
        moreover have sub2:"∀p ta. MCon ta (Memory st) p ∧ CompMemType (Memory st) len' (MTArray x
t) ta prnt p ⟶ p ≠ prnt"
          using MConPtrsMustBeSubLocs_CompMemType[OF mconPrnt] by auto
        moreover have ptrPrnt:"CompMemType (Memory st) len' (MTArray x t) (MTArray x t) prnt ptr"
using ptrDef in1 by auto
        ultimately have prtNotPrnt:"ptr ≠ prnt" by blast
        have sub3:"∀p ta. CompMemType (Memory st) x t ta ptr p ⟶ p ≠ prnt" using ptrPrnt sub2
ptrMC
          by (metis CompMemType_imps_CompMemJustType CompTypeRemainsMCon MConSubTypes mconPrnt)
        have "MCon (MTArray x t) (Memory st') ptr" using ptrMC sub3 prtNotPrnt
        proof (induction t arbitrary:x ptr)
          case (MTArray x1 t)
          then have xnotZero:"x ≠ 0" by fastforce
          have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                 (MTArray x1 t) (Memory st') (hash ptr (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray x1 t)
(Memory st') loc2 | MTValue Types ⇒ False)) ∧
            ((∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr (Memory st') =
None)"
            proof intros
              fix i2 assume i2Def:"i2<x"
              have "ptr ≠ prnt" using MTArray.prems by simp
              then have notL:"(hash ptr (ShowL_nat i2)) ≠ l" using ldef ShowLNatDot
                using hash_injective by blast
              then have sameAccess:"accessStore (hash ptr (ShowL_nat i2)) (Memory st') = accessStore
(hash ptr
                  (ShowL_nat i2)) (Memory st)" using nonLocChanged by auto
              then obtain ptr2 where ptr2Def: "accessStore (hash ptr (ShowL_nat i2)) (Memory st') =
Some (MPointer ptr2) ∧ MCon (MTArray x1 t) (Memory st) ptr2"
                using MTArray.prems(1) i2Def
                by (metis MConArrayPointers MCon_imps_sub_Mcon bot_nat_0.not_eq_extremum xnotZero)
              moreover have "∀p taa. CompMemType (Memory st) x1 t taa ptr2 p ⟶ p ≠ prnt " using
MTArray.prems ptr2Def i2Def
                by (metis sameAccess CompMemType.simps(2))
              moreover have "ptr2 ≠ prnt" using MTArray.prems ptr2Def i2Def
                by (metis sameAccess CompMemType.simps(2))
              ultimately show "case accessStore (hash ptr (ShowL_nat i2)) (Memory st') of None ⇒ False
                   | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False | MTValue typ ⇒
                       MCon (MTArray x1 t) (Memory st') (hash ptr (ShowL_nat i2)))
                   | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray
x1 t) (Memory st') loc2 | MTValue Types ⇒ False)"
                using MTArray.IH[of x1 ptr2] using ptr2Def by fastforce
              show " (∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr (Memory
st') = None "
              proof(cases "ptr = l")
                case True
                then show ?thesis using 11 by simp
              next
                case False
                then show ?thesis using nonLocChanged MTArray.prems(1)
                  using i2Def by fastforce
              qed
            qed
            then show ?case using MCon.simps(2)[of x "MTArray x1 t" "Memory st'" ptr] using xnotZero
by simp
        next
          case (MTValue x')
          then have xnotZero:"x ≠ 0" by fastforce
          have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
```

```
              | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                  (MTValue x') (Memory st') (hash ptr (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x')
(Memory st') loc2 | MTValue Types ⇒ False)) ∧
             ((∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr (Memory st') =
None)"
           proof(intros)
             fix i2 assume i2Def:"i2<x"
             have "(hash ptr (ShowL_nat i2)) ≠ l" using MTValue
               by (metis MCon_sub_MTVal_imps_val memoryvalue.distinct(1) i2Def lOrigin option.inject)
             then have "accessStore (hash ptr (ShowL_nat i2)) (Memory st') = accessStore (hash ptr
(ShowL_nat i2)) (Memory st)" using nonLocChanged by simp
             then show "case accessStore (hash ptr (ShowL_nat i2)) (Memory st') of None ⇒ False
      | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTValue
x') (Memory st') (hash ptr (ShowL_nat i2)))
      | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') (Memory st')
loc2 | MTValue Types ⇒ False)"
               using i2Def MTValue
               by (metis (lifting) CompTypeRemainsMCon mtypes.distinct(1) mtypes.simps(6)
memoryvalue.simps(5) Option.option.simps(5) linorder_less_linear mcon_accessStore not_less_zero
                   CompMemType.simps(1) MCon.simps(1))
             show "(∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr (Memory
st') = None"
             proof(cases "ptr = l")
               case True
               then show ?thesis using 11 by simp
             next
               case False
               then show ?thesis using nonLocChanged MTValue i2Def by auto
             qed
           qed
           then show ?case using MCon.simps(2)[of x "MTValue x'" "Memory st'" ptr] xnotZero by simp
         qed
         then show ?thesis
           by (simp add: ‹accessStore (hash prnt (ShowL_nat i1)) (Memory st') = accessStore (hash prnt
(ShowL_nat i1)) (Memory st)› ptrDef)
       qed
     qed

     then show ?thesis using lenNotZero p2 by simp
   qed

   have compmemst':"∀locs lens subTs. CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt
                       ∧ MCon (MTArray lens subTs) (Memory st) locs
                      ⟶ CompMemType (Memory (st')) lens subTs (MTArray len' arr') locs prnt"
   proof intros
     fix locs lens subTs
     assume asm1:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt ∧ MCon (MTArray
lens subTs) (Memory st) locs"
     then have asm5:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt" by blast
     have asm6:"MCon (MTArray lens subTs) (Memory st) locs" using asm1 by blast

     have mconl'':"MCon (MTArray len subT) (Memory st) l''" using lInfo(5) tParentType by simp


     then have a10:"CompMemType (Memory st') lens subTs (MTArray len' arr') locs prnt" using asm6
asm5
     proof(induction subTs arbitrary: lens locs)
       case (MTArray x11 x12)

       then obtain iIn lIn where iInDef:"iIn<lens ∧ accessStore (hash locs (ShowL_nat iIn)) (Memory
st) = Some (MPointer lIn)
                 ∧ (lIn = prnt ∧ MTArray x11 x12 = MTArray len' arr' ∨ CompMemType (Memory st) x11 x12
(MTArray len' arr') lIn prnt)"
```

```
        unfolding CompMemType.simps by blast
      have same2:" accessStore (hash locs (ShowL_nat iIn)) (Memory st) = accessStore (hash locs
(ShowL_nat iIn)) (Memory st)"
        using MConArrayPointers MTArray.prems(2) bot_nat_0.not_eq_extremum iInDef le_eq_less_or_eq
nonLocChanged not_less_zero option.discI by metis
      then have mcIn:"MCon (MTArray x11 x12) (Memory st) lIn" using same2 iInDef MTArray.prems
        by (metis MCon_imps_sub_Mcon)
      then show ?case
      proof(cases "lIn = prnt")
        case True
        have "prnt ≠ l" using ldef
          by (metis hash_inequality)
        moreover have "MTArray x11 x12 = MTArray len' arr'" using iInDef True
          by (metis CompMemType_imps_CompMemJustType MConSubTypes <MCon (MTArray x11 x12) (Memory st)
lIn> lsublocs mconPrnt CompMemJustType.simps(2))
        ultimately show ?thesis using True
          by (metis MConPtrsMustBeSubLocs iInDef lsublocs nonLocChanged CompMemType.simps(2))
      next
        case False
        then have "CompMemType (Memory st) x11 x12 (MTArray len' arr') lIn prnt" using iInDef by
blast
        then have cp:"CompMemType (Memory st') x11 x12 (MTArray len' arr') lIn prnt" using
MTArray.IH[of x11 lIn] mcIn t7 by simp
        have "(hash locs (ShowL_nat iIn)) ≠ l"
          by (metis CompMemType_imps_CompMemJustType CompMemTypes_asc MConSubTypes
memoryvalue.inject(2) iInDef lOrigin lsublocs mcIn mconlOld option.inject
              CompMemJustType.simps(2))
        then have "accessStore (hash locs (ShowL_nat iIn)) (Memory st) = accessStore (hash locs
(ShowL_nat iIn)) (Memory st')"
          by (simp add: nonLocChanged)
        then have "∃i<lens. ∃l. accessStore (hash locs (ShowL_nat i)) (Memory st') = Some (MPointer
l)
            ∧ (l = prnt ∧ MTArray x11 x12 = MTArray len' arr' ∨ CompMemType (Memory st') x11 x12
(MTArray len' arr') l prnt)"
          using False cp iInDef by metis
        then show ?thesis unfolding CompMemType.simps(2)
          using iInDef by blast
      qed
    next
      case (MTValue x2)
      then show ?case
        using compType by auto
    qed
    then show "CompMemType (Memory st') lens subTs (MTArray len' arr') locs prnt" by blast
  qed

  have "(type.Memory (MTArray len subT), Stackloc l') |∈| fmran (Denvalue env) ∧ accessStore l'
(Stack st) = Some (KMemptr l'')"
    using lInfo(2,3,4) tParentType by simp


  have subPNew_imp_old:"∀stl2. TypedMemSubPrefPtrs (Memory st') x t p stl2 ⟶ TypedMemSubPrefPtrs
(Memory st) x t p stl2"
  proof intros
    fix stl2
    assume in1:"TypedMemSubPrefPtrs (Memory st') x t p stl2"
    then show "TypedMemSubPrefPtrs (Memory st) x t p stl2 " using asf pNotPrnt
    proof(induction t arbitrary:x p)
      case (MTArray x1 t)
      obtain iin lin where linDef: "iin<x ∧ accessStore (hash p (ShowL_nat iin)) (Memory st') = Some
(MPointer lin)
                                  ∧ (lin = stl2 ∨ TypedMemSubPrefPtrs (Memory st') x1 t lin stl2)"
        using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
      have "(hash p (ShowL_nat iin)) ≠ l" using MTArray.prems(3) ldef ShowLNatDot
```

```
        by (metis hash_injective)
      then have sameAccess:" accessStore (hash p (ShowL_nat iin)) (Memory st') = accessStore (hash p
          (ShowL_nat iin)) (Memory st)" using nonLocChanged by simp
      then show ?case
      proof(cases "lin = stl2")
        case True
        then show ?thesis using MTArray.prems sameAccess linDef by force
      next
        case False
        then have "TypedMemSubPrefPtrs (Memory st') x1 t lin stl2" using linDef by simp
        moreover have "¬ TypedMemSubPrefPtrs (Memory st) x1 t lin prnt" using MTArray.prems(2)
linDef
          using sameAccess by auto
        moreover have " lin ≠ prnt" using MTArray.prems linDef
          using sameAccess by auto
        ultimately have "TypedMemSubPrefPtrs (Memory st) x1 t lin stl2"
          using MTArray.IH[of x1 lin] by blast
        then show ?thesis
          using linDef sameAccess by auto
      qed
    next
      case (MTValue x)
      then show ?case by simp
    qed
  qed

  have subPOld_imp_new_cmp:"∀stl2 x11 x12. CompMemType (Memory st) x t (MTArray x11 x12) p stl2 ⟶
CompMemType (Memory st') x t (MTArray x11 x12) p stl2"
  proof intros
    fix stl2 x11 x12
    assume in1:" CompMemType (Memory st) x t (MTArray x11 x12) p stl2"
    then show " CompMemType (Memory st') x t (MTArray x11 x12) p stl2" using asf pNotPrnt
    proof(induction t arbitrary:x p)
      case (MTArray x1 t)
      obtain iin lin where linDef: "iin<x ∧ accessStore (hash p (ShowL_nat iin)) (Memory st) = Some
(MPointer lin)
                                    ∧ (lin = stl2 ∧ MTArray x1 t = MTArray x11 x12 ∨ CompMemType
(Memory st) x1 t (MTArray x11 x12) lin stl2)"
        using MTArray.prems(1) unfolding CompMemType.simps by blast
      have "(hash p (ShowL_nat iin)) ≠ l" using MTArray.prems(3) ldef ShowLNatDot
        by (metis hash_injective)
      then have sameAccess:" accessStore (hash p (ShowL_nat iin)) (Memory st') = accessStore (hash p
          (ShowL_nat iin)) (Memory st)" using nonLocChanged by simp
      then show ?case
      proof(cases "lin = stl2")
        case True
        then show ?thesis using MTArray.prems sameAccess linDef
          by (metis (mono_tags, lifting) MTArray.IH CompMemType.simps(2) TypedMemSubPrefPtrs.simps(2))
      next
        case False
        then have "CompMemType (Memory st) x1 t (MTArray x11 x12) lin stl2" using linDef by simp
        moreover have "¬ TypedMemSubPrefPtrs (Memory st) x1 t lin prnt" using MTArray.prems(2)
linDef
          using sameAccess by auto
        moreover have " lin ≠ prnt" using MTArray.prems linDef
          using sameAccess by auto
        ultimately have "CompMemType (Memory st') x1 t (MTArray x11 x12) lin stl2"
          using MTArray.IH[of x1 lin] by blast
        then show ?thesis
          using linDef sameAccess by auto
      qed
    next
      case (MTValue x)
      then show ?case by simp
```

```
      qed
    qed

    have prntPNeg:"∀ stl2 x11 x12. ¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 p ⟶ ¬
TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 p"
    proof intros
      fix stl2 x11 x12
      assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 p"
      show "¬ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 p "
      proof
        assume in2:"TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 p"
        then show False using in1
        proof(induction x12 arbitrary:x11 stl2)
          case (MTArray x1 t)
          obtain iin lin where linDef: "iin<x11 ∧ accessStore (hash stl2 (ShowL_nat iin)) (Memory st) =
Some (MPointer lin)
                                    ∧ (lin = p ∨ TypedMemSubPrefPtrs (Memory st) x1 t lin p)"
            using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
          then show ?case
          proof(cases "(hash stl2 (ShowL_nat iin)) = l")
            case True
            then have "accessStore (hash stl2 (ShowL_nat iin)) (Memory st') = Some (MPointer p)" using
linDef 11 unfolding updateStore_def accessStore_def by simp
            then show ?thesis using MTArray.prems(2) linDef by auto
          next
            case False
            then have sameAccess:"accessStore (hash stl2 (ShowL_nat iin)) (Memory st) = accessStore
(hash stl2
                (ShowL_nat iin)) (Memory st')" using nonLocChanged by simp
            then show ?thesis
            proof(cases "lin = p")
              case True
              then show ?thesis using MTArray.prems(2) linDef sameAccess by auto
            next
              case False
              then have "TypedMemSubPrefPtrs (Memory st) x1 t lin p" using linDef by simp
              then show ?thesis using MTArray linDef sameAccess by auto
            qed
          qed
        next
          case (MTValue x)
          then show ?case by simp
        qed
      qed
    qed

    have subPNeg:"∀ stl2. ¬ TypedMemSubPrefPtrs (Memory st') x t p stl2 ⟶ ¬ TypedMemSubPrefPtrs
(Memory st) x t p stl2"
    proof intros
      fix stl2
      assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x t p stl2"
      show "¬ TypedMemSubPrefPtrs (Memory st) x t p stl2 "
      proof
        assume in2:" TypedMemSubPrefPtrs (Memory st) x t p stl2"
        then show False using in1 asf pNotPrnt
        proof(induction t arbitrary:x p)
          case (MTArray x1 t)
          obtain iin lin where linDef: "iin<x ∧ accessStore (hash p (ShowL_nat iin)) (Memory st) = Some
(MPointer lin)
                                    ∧ (lin = stl2 ∨ TypedMemSubPrefPtrs (Memory st) x1 t lin stl2)"
            using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
          then have "(hash p (ShowL_nat iin)) ≠ l" using ldef MTArray.prems ShowLNatDot
            by (metis hash_injective)
```

```
        then have sameAccess:"accessStore (hash p (ShowL_nat iin)) (Memory st) = accessStore (hash p
            (ShowL_nat iin)) (Memory st')" using nonLocChanged by simp
        then show ?thesis
        proof(cases "lin = stl2")
          case True
          then show ?thesis using MTArray.prems(2) linDef sameAccess by auto
        next
          case False
          then have "TypedMemSubPrefPtrs (Memory st) x1 t lin stl2" using linDef by simp
          then show ?thesis using MTArray linDef sameAccess
            by (metis TypedMemSubPrefPtrs.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed
  qed

  have prntPNeg_2:"∀stl2 x11 x12 dloc1. ¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 p ∧
TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 dloc1
                                  ⟶ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 dloc1 "
  proof intros
    fix stl2 x11 x12 dloc1
    assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 p ∧ TypedMemSubPrefPtrs (Memory st')
x11 x12 stl2 dloc1"
    then show "TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 dloc1"
    proof(induction x12 arbitrary:x11 stl2)
      case (MTArray x1 t)
      obtain iin lin where linDef: "iin<x11 ∧ accessStore (hash stl2 (ShowL_nat iin)) (Memory st') =
Some (MPointer lin)
                                    ∧ (lin = dloc1 ∨ TypedMemSubPrefPtrs (Memory st') x1 t lin dloc1)"
        using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
      then show ?case
      proof(cases "(hash stl2 (ShowL_nat iin)) = 1")
        case True
        then have "accessStore (hash stl2 (ShowL_nat iin)) (Memory st') = Some (MPointer p)" using
linDef 11 unfolding updateStore_def accessStore_def by simp
        then show ?thesis using MTArray.prems(1) linDef by auto
      next
        case False
        then have sameAccess:"accessStore (hash stl2 (ShowL_nat iin)) (Memory st) = accessStore (hash
stl2
            (ShowL_nat iin)) (Memory st')" using nonLocChanged by simp
        then show ?thesis
        proof(cases "lin = dloc1")
          case True
          then show ?thesis using MTArray.prems(1) linDef sameAccess by auto
        next
          case False
          then have "TypedMemSubPrefPtrs (Memory st') x1 t lin dloc1" using linDef by simp
          then show ?thesis using MTArray linDef sameAccess by auto
        qed
      qed
    next
      case (MTValue x)
      then show ?case by simp
    qed
  qed

  have prntPNeg_sublocs_comp:"∀stl2 x11 x12 dloc1 dt. ¬ TypedMemSubPrefPtrs (Memory st') x11 x12
stl2 p ∧ CompMemType (Memory st) x11 x12 dt stl2 dloc1
                                  ⟶ CompMemType (Memory st') x11 x12 dt stl2 dloc1 "
  proof intros
```

```
      fix stl2 x11 x12 dloc1 dt
      assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 p ∧ CompMemType (Memory st) x11 x12
dt stl2 dloc1"
      then show "CompMemType (Memory st') x11 x12 dt stl2 dloc1"
      proof(induction x12 arbitrary:x11 stl2)
        case (MTArray x1 t)
        obtain iin lin where linDef: "iin<x11 ∧ accessStore (hash stl2 (ShowL_nat iin)) (Memory st) =
Some (MPointer lin)
                                    ∧ (lin = dloc1 ∧ MTArray x1 t = dt ∨ CompMemType (Memory st) x1 t
dt lin dloc1)"
          using MTArray.prems(1) unfolding CompMemType.simps by blast
        then show ?case
        proof(cases "(hash stl2 (ShowL_nat iin)) = l")
          case True
          then have "accessStore (hash stl2 (ShowL_nat iin)) (Memory st') = Some (MPointer p)" using
linDef 11 unfolding updateStore_def accessStore_def by simp
          then show ?thesis using MTArray.prems(1) linDef by auto
        next
          case False
          then have sameAccess:"accessStore (hash stl2 (ShowL_nat iin)) (Memory st) = accessStore (hash
stl2
              (ShowL_nat iin)) (Memory st')" using nonLocChanged by simp
          then show ?thesis
          proof(cases "lin = dloc1")
            case True
            then show ?thesis using MTArray.prems(1) linDef sameAccess
              using MTArray.IH by auto
          next
            case False
            then have "CompMemType (Memory st) x1 t dt lin dloc1" using linDef by simp
            then show ?thesis using MTArray linDef sameAccess by auto
          qed
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed

    have pPrntsLim:"∀x11 x12 stl2. CompMemType (Memory st) x11 x12 (MTArray x t) stl2 p ∧ accessStore
l (Memory st) ≠
        Some (MPointer p) ∧ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 p ∧
  MCon (MTArray x11 x12) (Memory st) stl2 ⟶ CompMemType (Memory st') x11 x12 (MTArray x t) stl2 p"
    proof(intros)
      fix x11 x12 stl2
      assume "CompMemType (Memory st) x11 x12 (MTArray x t) stl2 p ∧
     accessStore l (Memory st) ≠ Some (MPointer p) ∧ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 p ∧
MCon (MTArray x11 x12) (Memory st) stl2"
      then show "CompMemType (Memory st') x11 x12 (MTArray x t) stl2 p"
      proof(induction x12 arbitrary: x11 stl2)
        case (MTArray x1 x12)
        obtain inI inL where inDef:"inI<x11 ∧ accessStore (hash stl2 (ShowL_nat inI)) (Memory st) =
Some (MPointer inL)
                           ∧ (inL = p ∨ TypedMemSubPrefPtrs (Memory st) x1 x12 inL p)"
          using MTArray.prems unfolding TypedMemSubPrefPtrs.simps by blast
        then show ?case
        proof(cases "inL = p")
          case True
          then show ?thesis
            by (metis MTArray.prems inDef
                existingLocation_imps_allLocs_same nonLocChanged CompMemType.simps(2))
        next
          case False
          then have "TypedMemSubPrefPtrs (Memory st) x1 x12 inL p" using inDef by auto
```

```
          moreover have "accessStore l (Memory st) ≠ Some (MPointer p)" using MTArray.prems by simp
          moreover have "MCon (MTArray x1 x12) (Memory st) inL" using MTArray.prems inDef
            using MconSameTypeSameAccessWithTyping by blast
          moreover have "CompMemType (Memory st) x1 x12 (MTArray x t) inL p"
            by (meson False MTArray.prems existingLocation_imps_allLocs inDef)
          ultimately have "CompMemType (Memory st') x1 x12 (MTArray x t) inL p" using MTArray.IH[of x1
inL] by blast
          moreover have "accessStore (hash stl2 (ShowL_nat inI)) (Memory st) = accessStore (hash stl2
(ShowL_nat inI)) (Memory st')"
          proof -
            obtain nn :: "nat ⇒ String.literal ⇒ String.literal ⇒ nat" where
              f1: "∀X1 X4 X5. (∃X7. hash X4 (ShowL_nat X7) = X5 ∧ X7 < X1) ⟶ hash
                  X4 (ShowL_nat (nn X1 X4 X5)) = X5 ∧ nn X1 X4 X5 < X1"
              by moura
            obtain nna :: "String.literal ⇒ String.literal ⇒ nat ⇒ mtypes ⇒ nat" and nnb
                :: "String.literal ⇒ String.literal ⇒ nat ⇒ mtypes ⇒ nat" where
              f2: "∀l la n m. (TypedMemSubPref l la (MTArray n m) ∨ (∀na. hash la (ShowL_nat na) ≠ l
          ∧ ¬ TypedMemSubPref l (hash la (ShowL_nat na)) m ∨ ¬ na < n)) ∧ ((hash la (ShowL_nat(nna l la
n m)) = l
           ∨ TypedMemSubPref l (hash la (ShowL_nat(nna l la n m))) m) ∧ nna l la n m < n ∨ ¬
TypedMemSubPref l la (MTArray n m))"
              using TypedMemSubPref.simps(2) by moura
            obtain nnc :: "nat ⇒ String.literal ⇒ String.literal ⇒ nat" where
              f3: "∀s n t m l la. (CompMemType s n (MTValue t) m l la ∨ (∀na. hash l (ShowL_nat na) ≠
la ∨ ¬ na < n)
        ∨ MTValue t ≠ m) ∧ ((hash l (ShowL_nat(nnc n l la)) = la ∧ nnc n l la < n) ∧ MTValue t = m ∨ ¬
CompMemType s n (MTValue t) m l la)"
              using f1 CompMemType.simps(1) by moura
            obtain ll :: "String.literal ⇒ nat ⇒ mtypes ⇒ (mtypes, memoryvalue) typedstore ⇒
String.literal"
                   and lla :: "String.literal ⇒ nat ⇒ mtypes ⇒ memoryvalue store ⇒ String.literal"
where
                f4: "TypedMemSubPref (ll inL x1 x12 (Memory st)) inL (MTArray x1 x12) ∨ TypedMemSubPref
p inL (MTArray x1 x12)"
              by (meson CompMemType_imps_TypedMemSubPrefPtrs ‹CompMemType (Memory st) x1 x12 (MTArray x
t) inL p› selfPoint_imps_TypedMemSubPref)
            obtain tt :: "mtypes ⇒ types" and nnd :: "mtypes ⇒ nat" and mm :: "mtypes ⇒ mtypes"
where
              "x12 = MTArray (nnd x12) (mm x12) ∨ x12 = MTValue (tt x12) ∨ 0 = x1"
              by (metis (no_types) ‹MCon (MTArray x1 x12) (Memory st) inL› bot_nat_0.not_eq_extremum
mcon_accessStore)
            then show ?thesis
              using f4 f3 f2 by (metis CompMemType_imps_CompMemJustType MConSubTypes
memoryvalue.inject(2) ‹CompMemType (Memory st)
                  x1 x12 (MTArray x t) inL p› ‹MCon (MTArray x1 x12) (Memory st) inL› inDef lOrigin
less_nat_zero_code
                  mconlOld nonLocChanged option.inject CompMemJustType.simps(2) CompMemType.simps(2))
          qed
          ultimately show ?thesis using inDef unfolding CompMemType.simps by metis
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed

    qed

    have getToPrntNew_imps_Old:"∀stl2 x11 x12. TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt ∧
prnt ≠ stl2
        ⟶ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
    proof intros
      fix stl2 x11 x12
      assume "TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt ∧ prnt ≠ stl2"
      then show "TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
```

```
      proof(induction x12 arbitrary:x11 stl2)
        case (MTArray x1 x12)
        obtain ii ll where llDef:"ii<x11 ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = Some
(MPointer ll)
                                  ∧ (ll = prnt ∨ TypedMemSubPrefPtrs (Memory st') x1 x12 ll prnt)"
          using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast

        then show ?case
        proof(cases "(hash stl2 (ShowL_nat ii)) = l")
          case True
          then have "stl2 = prnt" using ldef ShowLNatDot
            using hash_injective by blast
          then show ?thesis using MTArray.prems(1) by auto
        next
          case False
          then have same:" accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = accessStore (hash stl2
              (ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
          then show ?thesis
          proof(cases "ll = prnt")
            case True
            then show ?thesis using same llDef by force
          next
            case False
            then have "TypedMemSubPrefPtrs (Memory st') x1 x12 ll prnt" using llDef by simp
            then show ?thesis using MTArray.IH[of x1 ll] same False llDef by force
          qed
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed

    have prntPrntNeg:"∀stl2 x11 x12. ¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt ∧ prnt ≠
stl2
        ⟶ ¬ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
    proof intros
      fix stl2 x11 x12
      assume in1:"¬ TypedMemSubPrefPtrs (Memory st') x11 x12 stl2 prnt ∧ prnt ≠ stl2"
      show "¬ TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt "
      proof
        assume in2:"TypedMemSubPrefPtrs (Memory st) x11 x12 stl2 prnt"
        then show False using in1
        proof(induction x12 arbitrary:x11 stl2)
          case (MTArray x1 t)
          obtain iIn lin where linDef: "iIn<x11 ∧ accessStore (hash stl2 (ShowL_nat iIn)) (Memory st) =
Some (MPointer lin)
                                    ∧ (lin = prnt ∨ TypedMemSubPrefPtrs (Memory st) x1 t lin prnt)"
            using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by blast
          then show ?case
          proof(cases "(hash stl2 (ShowL_nat iIn)) = l")
            case True
            then have "stl2 = prnt" using ldef ShowLNatDot
              using hash_injective by blast
            then show ?thesis using MTArray.prems(2) linDef by simp
          next
            case False
            then have sameAccess:"accessStore (hash stl2 (ShowL_nat iIn)) (Memory st) = accessStore
(hash stl2
                (ShowL_nat iIn)) (Memory st')" using nonLocChanged by simp
            then show ?thesis
            proof(cases "lin = prnt")
              case True
              then show ?thesis using MTArray.prems(2) linDef sameAccess by auto
```

```
            next
              case False
              then have "TypedMemSubPrefPtrs (Memory st) x1 t lin prnt" using linDef by simp
              then show ?thesis using MTArray linDef sameAccess by auto
            qed
          qed
        next
          case (MTValue x)
          then show ?case by simp
        qed
      qed
    qed


    have prntLim3Neg:"∀x11' x12' stl2 stl1. ¬TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1 ∧ ¬
TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1 ∧
          stl2 ≠ prnt ∧ ( ∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt ⟶
newt = MTArray len' arr')
          ⟶ ¬TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 stl1"
    proof intros
      fix x11' x12' stl2 stl1
      assume "¬TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1 ∧
     ¬ TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1 ∧
     stl2 ≠ prnt ∧ (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt ⟶ newt
= MTArray len' arr')"
      then have in1:"¬TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1"
          and "¬ TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1"
          and "stl2 ≠ prnt"
          and " (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt ⟶ newt =
MTArray len' arr')" by simp+
      then show "¬TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 stl1"
      proof(induction x12' arbitrary:x11' stl2)
        case (MTArray x1 x12')
        then have iiNeg: "∀i<x11'. ∃l. accessStore (hash stl2 (ShowL_nat i)) (Memory st') ≠ Some
(MPointer l)
                                        ∨ (l ≠ stl1 ∧ ¬TypedMemSubPrefPtrs (Memory st') x1 x12' l
stl1)"
          unfolding TypedMemSubPrefPtrs.simps by blast
        show ?case
        proof
          assume "TypedMemSubPrefPtrs (Memory st) x11' (MTArray x1 x12') stl2 stl1"
          then obtain ii ll where iiDef: "ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st)
= Some (MPointer ll)
                    ∧ (ll = stl1 ∨ TypedMemSubPrefPtrs (Memory st) x1 x12' ll stl1)"
            unfolding TypedMemSubPrefPtrs.simps by blast
          then have "(hash stl2 (ShowL_nat ii)) ≠ l" using ldef hash_injective ShowLNatDot
MTArray.prems by blast
          then have same:" accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = accessStore (hash stl2
              (ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
          show False
          proof(cases "ll = stl1")
            case True
            then show ?thesis using iiDef same iiNeg
              using MTArray.prems(1) by auto
          next
            case False
            then have s:"TypedMemSubPrefPtrs (Memory st) x1 x12' ll stl1" using iiDef by simp
            have llNotPrnt:"ll ≠ prnt"
            proof
              assume *:"ll = prnt"
              then have "MTArray x1 x12' = MTArray len' arr'" using MTArray.prems(4) iiDef by auto
              then show False using s MTArray.prems(2) * by simp
            qed
            then show ?thesis using same MTArray.IH[OF ] iiDef MTArray.prems(4)
```

```
            by (metis MTArray.prems(1,2) CompMemType.simps(2) TypedMemSubPrefPtrs.simps(2))
        qed
      qed

    next
      case (MTValue x)
      then show ?case by simp
    qed
  qed

  have prntLim4:"∀x11' x12' stl2 stl1 dt. CompMemType (Memory st) x11' x12' dt stl2 stl1 ∧ ¬
TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1 ∧
      stl2 ≠ prnt ∧ ( ∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt
          ⟶ newt = MTArray len' arr') ∧ MCon (MTArray x11' x12') (Memory st) stl2
      ⟶ CompMemType (Memory st') x11' x12' dt stl2 stl1"
  proof intros
    fix x11' x12' stl2 stl1 dt
    assume "CompMemType (Memory st) x11' x12' dt stl2 stl1 ∧
   ¬ TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1 ∧
   stl2 ≠ prnt ∧ (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt ⟶
       newt = MTArray len' arr') ∧ MCon (MTArray x11' x12') (Memory st) stl2"
    then have in1:"CompMemType (Memory st) x11' x12' dt stl2 stl1"
        and "¬ TypedMemSubPrefPtrs (Memory st) len' arr' prnt stl1"
        and "stl2 ≠ prnt"
        and " (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = prnt ⟶ newt =
MTArray len' arr')"
        and "MCon (MTArray x11' x12') (Memory st) stl2" by blast+
    then show "CompMemType (Memory st') x11' x12' dt stl2 stl1"
    proof(induction x12' arbitrary:x11' stl2)
      case (MTArray x1 x12')
      then obtain ii ll where iiDef: "ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st) =
Some (MPointer ll)
                ∧ (ll = stl1 ∧ MTArray x1 x12' = dt ∨ CompMemType (Memory st) x1 x12' dt ll stl1)"
        unfolding CompMemType.simps by blast
      then have "(hash stl2 (ShowL_nat ii)) ≠ l" using ldef hash_injective ShowLNatDot MTArray.prems
by blast
      then have same:" accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = accessStore (hash stl2
(ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
      then show ?case
      proof(cases "ll = stl1")
        case True
        then have "MTArray x1 x12' = dt" using iiDef True
          by (metis MTArray.prems(1,5) existingLocation_imps_allLocs_same)
        then show ?thesis using iiDef same True by force
      next
        case False
        then have s:"CompMemType (Memory st) x1 x12' dt ll stl1" using iiDef by simp
        have llNotPrnt:"ll ≠ prnt"
        proof
          assume *:"ll = prnt"
          then have "MTArray x1 x12' = MTArray len' arr'" using MTArray.prems(4) iiDef same by auto
          then show False using s MTArray.prems(2) * same
            by (simp add: CompMemType_imps_TypedMemSubPrefPtrs)
        qed
        have "MCon (MTArray x1 x12') (Memory st) ll" using MTArray.prems(5) iiDef
          using MconSameTypeSameAccessWithTyping by blast
        then have "CompMemType (Memory st') x1 x12' dt ll stl1"
          using MTArray.IH[OF s MTArray.prems(2) llNotPrnt] iiDef MTArray.prems(4,5) by auto
        then show ?thesis using same iiDef by auto
      qed
    next
      case (MTValue x)
      then show ?case by simp
    qed
```

```
    qed

    have prntLim5:"∀x11' x12' stl2 dloc1 dt. TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 dloc1
                    ∧ (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = dloc1 ⟶
newt = dt)
                   ∧ stl2 ≠ prnt ∧ ¬ TypedMemSubPrefPtrs (Memory st') len' arr' prnt dloc1
                  ∧ (∀new newt. CompMemType (Memory st') x11' x12' newt stl2 new ∧ new = prnt ⟶ newt
= MTArray len' arr')
                   ⟶ CompMemType (Memory st') x11' x12' dt stl2 dloc1"
    proof intros
      fix x11' x12' stl2 dloc1 dt
      assume " TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 dloc1 ∧
     (∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = dloc1 ⟶ newt = dt) ∧
     stl2 ≠ prnt ∧
     ¬ TypedMemSubPrefPtrs (Memory st') len' arr' prnt dloc1 ∧ (∀new newt. CompMemType (Memory st')
x11' x12' newt stl2
         new ∧ new = prnt ⟶ newt = MTArray len' arr')"
      then have " TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 dloc1" and
        "(∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = dloc1 ⟶ newt = dt)"
          and "stl2 ≠ prnt"
          and "¬ TypedMemSubPrefPtrs (Memory st') len' arr' prnt dloc1"
          and "(∀new newt. CompMemType (Memory st') x11' x12' newt stl2 new ∧ new = prnt ⟶ newt =
MTArray len' arr')" by blast+
      then show "CompMemType (Memory st') x11' x12' dt stl2 dloc1"
      proof(induction x12' arbitrary: x11' stl2)
        case (MTArray x1 x12')
        obtain ii ll where iiDef:"ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = Some
(MPointer ll)
                                  ∧ (ll = dloc1 ∨ TypedMemSubPrefPtrs (Memory st') x1 x12' ll dloc1)"
          using MTArray.prems(1) unfolding TypedMemSubPrefPtrs.simps by auto
        then have "(hash stl2 (ShowL_nat ii)) ≠ l" using ldef hash_injective ShowLNatDot
MTArray.prems(3) by blast
        then have same:"accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = accessStore (hash stl2
(ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
        then show ?case
        proof(cases "ll = dloc1")
          case True
          then have "(MTArray x1 x12') = dt" using iiDef MTArray.prems(2) same by auto
          then show ?thesis using iiDef True by force
        next
          case False
          then have cc1:"TypedMemSubPrefPtrs (Memory st') x1 x12' ll dloc1" using iiDef by simp
          have cc2:"∀new newt. CompMemType (Memory st) x1 x12' newt ll new ∧ new = dloc1 ⟶ newt =
dt" using iiDef MTArray.prems(2) same by auto
          have cc3:"ll ≠ prnt"
          proof
            assume *:"ll = prnt"
            then have "(MTArray x1 x12') = MTArray len' arr'" using MTArray.prems(5) iiDef by auto
            then show False using MTArray.prems(4) cc1 * by simp
          qed
          have "∀new newt. CompMemType (Memory st') x1 x12' newt ll new ∧ new = prnt ⟶ newt =
MTArray len' arr'" using MTArray.prems(5) same iiDef by force
          then have "CompMemType (Memory st') x1 x12' dt ll dloc1"
            using MTArray.IH[OF cc1 cc2 cc3 MTArray.prems(4)] MTArray.prems iiDef same by blast
          then show ?thesis using iiDef same by auto
        qed
      next
        case (MTValue x)
        show ?case using MTValue(1,2) by auto
      qed
    qed

    have prntLim6:"∀x11' x12' stl2 dloc1. ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt∧
TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 dloc1∧ stl2 ≠ prnt
```

```
          ⟶ TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 dloc1"
    proof intros
      fix x11' x12' stl2 dloc1
      assume " ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt ∧ TypedMemSubPrefPtrs (Memory
st') x11' x12' stl2 dloc1 ∧ stl2 ≠ prnt"
      then show "TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 dloc1 "
      proof(induction x12' arbitrary:x11' stl2)
        case (MTArray x1 x12')
        obtain ii ll where iiDef:"ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st') = Some
(MPointer ll) ∧
                                    (ll = dloc1 ∨ TypedMemSubPrefPtrs (Memory st') x1 x12' ll
dloc1)"
          using MTArray.prems unfolding TypedMemSubPrefPtrs.simps by blast
        then have "(hash stl2 (ShowL_nat ii)) ≠ l" using ldef hash_injective ShowLNatDot MTArray by
blast
        then have same:"accessStore (hash stl2 (ShowL_nat ii)) (Memory st') =accessStore (hash stl2
(ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
        then show ?case
        proof(cases "ll = dloc1")
          case True
          then show ?thesis using same iiDef by force
        next
          case False
          then show ?thesis using iiDef same MTArray by auto
        qed
      next
        case (MTValue x)
        then show ?case by auto
      qed
    qed

    have prntLim7:"∀x11' x12' stl2 stl1 dt.(∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new
∧ new = stl1 ⟶ newt = dt)
                  ∧ ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt ∧ stl2 ≠ prnt
                  ∧ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1
                   ⟶ CompMemType (Memory st') x11' x12' dt stl2 stl1"
    proof intros
      fix x11' x12' stl2 stl1 dt
      assume "(∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = stl1 ⟶ newt = dt)
∧
     ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt ∧ stl2 ≠ prnt ∧ TypedMemSubPrefPtrs
(Memory st') x11' x12' stl2 stl1"
      then have "(∀new newt. CompMemType (Memory st) x11' x12' newt stl2 new ∧ new = stl1 ⟶ newt =
dt)"
          and "¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt"
          and "stl2 ≠ prnt"
          and "TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1" by blast+
      then show "CompMemType (Memory st') x11' x12' (dt) stl2 stl1"
      proof(induction x12' arbitrary:x11' stl2)
        case (MTArray x1 x12')
        then obtain ii ll where iiDef:"ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st') =
Some (MPointer ll)
                                    ∧ (ll = stl1 ∨ TypedMemSubPrefPtrs (Memory st') x1 x12' ll stl1)"
          unfolding TypedMemSubPrefPtrs.simps by blast
        then have "(hash stl2 (ShowL_nat ii)) ≠ l" using MTArray.prems ldef ShowLNatDot hash_injective
by blast
        then have same:"accessStore (hash stl2 (ShowL_nat ii)) (Memory st') =accessStore (hash stl2
(ShowL_nat ii)) (Memory st)" using nonLocChanged by simp
        then show ?case
        proof(cases "ll = stl1")
          case True
          then show ?thesis using same MTArray.prems(1) iiDef by force
        next
          case False
```

```
        then show ?thesis using same MTArray iiDef
          by (metis (no_types, opaque_lifting) CompMemType.simps(2) TypedMemSubPrefPtrs.simps(2))
      qed
    next
      case (MTValue x)
      then show ?case by auto
    qed
  qed

  have prntLim8:"∀x11' x12' stl2 stl1. ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1 ∧
              ¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 prnt ∧ stl2 ≠ prnt
              ⟶ ¬ TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 stl1"
  proof intros
    fix x11' x12' stl2 stl1
    assume "¬ TypedMemSubPrefPtrs (Memory st') x11' x12' stl2 stl1 ∧ ¬ TypedMemSubPrefPtrs (Memory
st') x11' x12' stl2 prnt ∧ stl2 ≠ prnt"
    then show "¬ TypedMemSubPrefPtrs (Memory st) x11' x12' stl2 stl1"
    proof(induction x12' arbitrary:x11' stl2)
      case (MTArray x1 x12')

      show ?case
      proof
        assume *:"TypedMemSubPrefPtrs (Memory st) x11' (MTArray x1 x12') stl2 stl1"
        then obtain ii ll where iiDef:"ii<x11' ∧ accessStore (hash stl2 (ShowL_nat ii)) (Memory st) =
Some (MPointer ll)
                                    ∧ (ll = stl1 ∨ TypedMemSubPrefPtrs (Memory st) x1 x12' ll
stl1)"
          unfolding TypedMemSubPrefPtrs.simps by blast
        then have " (hash stl2 (ShowL_nat ii)) ≠ l" using ldef hash_injective ShowLNatDot
MTArray.prems by blast
        then have same:"accessStore (hash stl2 (ShowL_nat ii)) (Memory st) = accessStore (hash stl2
            (ShowL_nat ii)) (Memory st')" using nonLocChanged by simp
        show False
        proof(cases "ll = stl2")
          case True
          then show ?thesis using iiDef same MTArray.prems
            using MTArray.IH by force
        next
          case False
          then show ?thesis using iiDef same MTArray by auto
        qed
      qed
    next
      case (MTValue x)
      then show ?case by simp
    qed
  qed


  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 11 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    then show "compPointers (Stack st') (Denvalue env)" using sameStack sameStorage by simp
  next
    show "safeContract (Accounts st') (Storage st')" using sameStorage 11 2(1) unfolding
TypeSafe_def safeContract_def by auto
  next
    show "balanceTypes (Accounts st')" using 11 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
```

```
      then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
    next
      show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore
loc (Stack st) = None) ∧
            (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
        using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
      have **:"Toploc (Stack st) = Toploc (Stack st')" using 11 unfolding updateStore_def by auto
      show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
      proof intros

        fix tloc loc
        assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
        then show "accessStore loc (Stack st') = None" using *
          by (simp add: sameStack)
      next
        fix loc y
        assume h1:" accessStore loc (Stack st') = Some y"
        then show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
          by (metis sameStack)
      qed
    next
      show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
    next
      have a12:"Toploc (Memory st) = Toploc (Memory st')" using 11 unfolding updateStore_def by auto
      have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp
      have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
      show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have b20:"Toploc (Memory st) ≤ tloc" using a12 tloc a12
          by force

        then show "accessStore loc (Memory st') = None "
          by (metis a15 b10 lOrigin lessThanSome_imps_Locs lessThanTopLocs_def linorder_not_less
nonLocChanged)
      next
        fix loc y
        assume "accessStore loc (Memory st') = Some y "
        then show "∃tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
          by (metis a12 a15 lOrigin lessThanTopLocs_def nonLocChanged)
      qed
    next
      show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
        unfolding typeCompat_def
      proof intros
        fix tLook lLook
        assume inDen:" (tLook, lLook) |∈| fmran (Denvalue env)"
        show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd tLook lLook"
        proof(cases lLook)
          case (Stackloc x1)
          then show ?thesis
          proof(cases "accessStore x1 (Stack st')")
            case None
            then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc by force
          next
            case (Some a)
            then show ?thesis
```

```
          proof(cases a)
            case (KValue x1')
            then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by force
          next
            case (KCDptr x2)
            then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by force
          next
            case (KMemptr x3)
            then obtain struct where stT: "tLook = type.Memory struct" using 2(1) unfolding
TypeSafe_def typeCompat_def using sameStorage inDen sameStack Stackloc Some
              by (cases tLook; fastforce)
            have mcOld:"MCon struct (Memory st) x3"
              using sameStorage inDen sameStack Stackloc Some KMemptr by (metis 2(1) sameMemTSafe
stT)
            have tps:"(case struct of MTArray len arr ⇒ ∀i<len. accessTypeStore (hash x3 (ShowL_nat
i)) (Memory st) = Some arr
                        | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def
              using inDen sameStack Stackloc Some KMemptr stT by simp
            have nc:"∀locs. locs ≠ l ⟶ accessStore locs (Memory st) = accessStore locs (Memory
st')" using nonLocChanged by simp
            have subt:"subPrefixStructuralConsistency (Memory st)" using 2(1) unfolding TypeSafe_def
by simp
            have "accessTypeStore l (Memory st) = Some (MTArray x t)" using lInfo 11 by auto
            then have "MCon struct (Memory st') x3"
              using 11(4) cpm2m_singleLChange[OF mcOld tps _ _ _ mcPNew subt nc ] lOrigin
              by simp
            then show ?thesis using sameStorage inDen sameStack Stackloc Some KMemptr stT by simp
          next
            case (KStoptr x4)
            then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage
inDen sameStack Stackloc Some by (cases tLook; force)
          qed
        qed
      next
        case (Storeloc x2)
        then show ?thesis using 2(1) unfolding TypeSafe_def typeCompat_def using sameStorage inDen
          by (metis denvalue.simps(6))
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd "
using sameStack by simp
  next
    have "Accounts st'= Accounts st" using 11 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 11 unfolding
fullyInitialised_def by auto
  next
    have sameACCT:"∀locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
      using 11 lInfo unfolding accessTypeStore_def updateStore_def by auto

    show "denvalueTypeCorrectness env (Stack st') (Memory st') "
      unfolding denvalueTypeCorrectness_def
    proof(intros)
      fix t2 l2 ptr_loc sub_loc
      assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env) ∧ accessStore l2 (Stack st')
= Some (KMemptr ptr_loc)"

      show "case t2 of
     MTArray len arr ⇒
     (∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
   | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
      proof -
```

```
        have sameACC:"accessStore l2 (Stack st') = accessStore l2 (Stack st)"
          using * 11 unfolding accessStore_def updateStore_def by simp
        then have mcOld:"MCon t2 (Memory st) ptr_loc" using * by (metis 2(1) sameMemTSafe)
        then have old:"(case t2 of
                        MTArray len arr ⇒
       (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)

     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
          using sameACC 2(1) * TypeSafe_def denvalueTypeCorrectness_def by auto

        have inDenvalue:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env)" using * by simp

        then show ?thesis
        proof(cases t2)
          case (MTArray x11 x12)
          then have mcOld:"MCon (MTArray x11 x12) (Memory st) ptr_loc" using mcOld by blast

          then have conc1:"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some
x12)"
            using sameACCT old MTArray by simp
          then show ?thesis using MTArray
            using lOrigin nonLocChanged old sameACCT by auto
        next
          case (MTValue x2)
          have "accessTypeStore ptr_loc (Memory st') = Some (MTValue x2)"
            using MTValue old sameACCT by auto
          then show ?thesis using MTValue by auto
        qed
      qed
    qed
  next
    have oldSub:"subPrefixStructuralConsistency (Memory st)"
      using 2(1) unfolding TypeSafe_def by simp
    show "subPrefixStructuralConsistency (Memory st')"
      unfolding subPrefixStructuralConsistency_def
    proof intros
      fix locs tp
      assume in1:" accessTypeStore locs (Memory st') = Some tp "
      have sameTy:"∀ locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st') "
        using 11 unfolding updateStore_def accessTypeStore_def by auto
      show "case accessStore locs (Memory st') of None ⇒ False
            | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st') locs ∧ tp = MTValue val ∧
accessTypeStore locs (Memory st') = Some tp
            | Some (MPointer p) ⇒
              ∃ len arr.
                 MCon tp (Memory st') p ∧
                 tp = MTArray len arr ∧
                 (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some arr)"
      proof(cases "locs = l")
        case True
        have actO:"accessTypeStore l (Memory st) = Some (MTArray x t)"
          using lInfo 11 nonLocChanged by auto
        then have tpDef:"tp = MTArray x t"
          using sameTy True in1 by simp
        then have ptr:"accessStore locs (Memory st') = Some (MPointer p)"
          using 11(4) True by simp
        moreover have " MCon tp (Memory st') p"
          using tpDef mcPNew by fastforce
        moreover have "(∀ i<x. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some t)"
          using sameTy pTAccessOld by simp
        ultimately show ?thesis using True tpDef ptr by auto
      next
        case False
        then have sameACC:"accessStore locs (Memory st) = accessStore locs (Memory st')"
```

```
          using nonLocChanged by simp
        then have in3:"(case accessStore locs (Memory st) of None ⇒ False | Some (MValue v)
    ⇒ ∃val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs (Memory st) = Some tp
      | Some (MPointer p) ⇒
        ∃len arr.
           MCon tp (Memory st) p ∧
           tp = MTArray len arr ∧
           (∀i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some arr) )"
          using oldSub in1 sameTy unfolding subPrefixStructuralConsistency_def by fastforce
        then show ?thesis
        proof(cases "accessStore locs (Memory st)")
          case None
          then show ?thesis using in3 sameACC by simp
        next
          case (Some a)
          then show ?thesis
          proof(cases a)
            case (MValue x1)
            then obtain val where " MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore
locs (Memory st) = Some tp"
              using in3 Some by auto
            moreover have "MCon tp (Memory st') locs" using sameACC calculation by auto
            ultimately show ?thesis using sameACC Some MValue sameTy
              by (auto split:option.splits)
          next
            case (MPointer x2)
            have accl:"accessTypeStore l (Memory st) = Some (MTArray x t)"
              using lInfo 11 nonLocChanged by auto

            then obtain len arr where s:"MCon tp (Memory st) x2 ∧ tp = MTArray len arr ∧
                      (∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) (Memory st) = Some arr)"
              using Some in3 MPointer by auto
            then have mcM:"MCon tp (Memory st) x2" by auto
            have lTyp:"accessTypeStore l (Memory st) = Some (MTArray x t)"
              using lInfo 11 by simp
            then have "MCon tp (Memory st') x2"
              using cpm2m_singleLChange[OF mcM _ _ _ lTyp mcPNew oldSub]
                11(4) accl s lOrigin nonLocChanged by auto
            moreover have "(∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) (Memory st') = Some arr)"
              using s sameTy by simp

            ultimately show ?thesis using Some MPointer sameACC sameTy s
              by (auto split:option.splits)
          qed
        qed
      qed
    qed

  next

    have old:"(∀locs. (∃t. accessStore locs (Memory st) = Some t) = (∃tt. accessTypeStore locs
(Memory st) = Some tt))"
      using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    have sameACCT:"∀locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
      using 11 lInfo unfolding accessTypeStore_def updateStore_def by auto

    show "SomeValSomeTyp (Memory st')"unfolding SomeValSomeTyp_def
    proof intros
      fix locs
      show "(∃t. accessStore locs (Memory st') = Some t) = (∃tt. accessTypeStore locs (Memory st') =
Some tt) "
      proof(cases "locs = l")
        case True
        then have acc:"accessStore l (Memory st') = Some (MPointer p)"
```

```
          using 11 by simp
        then have "∃v. accessStore l (Memory st) = Some v" using lOrigin by auto
        then have "∃t. accessTypeStore l (Memory st) = Some t"
          using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by simp
        then show ?thesis using acc sameACCT True by simp
      next
        case False
        then show ?thesis using old sameACCT nonLocChanged by simp
      qed
    qed
  next
    fix locs t
    assume a1:"accessTypeStore locs (Memory st) = Some t"
    have old:"(∀locs. (∃t. accessStore locs (Memory st) = Some t) = (∃tt. accessTypeStore locs
(Memory st) = Some tt))"
      using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    have sameACCT:"∀locs. accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
      using 11 lInfo unfolding accessTypeStore_def updateStore_def by auto

    show "accessTypeStore locs (Memory st') = Some t"
    proof(cases "locs = l")
      case True
      then have "∃t. accessTypeStore l (Memory st) = Some t"
        using 2(1) a1 unfolding TypeSafe_def SomeValSomeTyp_def by simp
      then show ?thesis using sameACCT True a1 by simp
    next
      case False
      then show ?thesis using old sameACCT nonLocChanged a1 by simp
    qed
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs
(Memory st') = Some (MPointer v')"
      using accL nonLocChanged by auto
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      by (metis memoryvalue.distinct(1) lOrigin nonLocChanged option.inject)
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      by (metis lOrigin nonLocChanged option.distinct(1))
  next
    have a12:"Toploc (Memory st) = Toploc (Memory st')" using 11 unfolding updateStore_def by auto
    then show "Toploc (Memory st) ≤ Toploc (Memory st')" by simp
  qed
 qed
qed
```

Assignment case 12: a storage array assigned to a memory variable on the stack: the array is copied into fresh memory.

```
lemma TypeSafe_Assign_12:
 assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
   and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
   and "fullyInitialised env (Accounts st) (Stack st)"
   and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
       ex) env cd st) = Normal ((KStoptr p, type.Storage (STArray x t)), g)"
   and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStackloc l, type.Memory t'),g')"
   and "cps2mTypeCompatible (STArray x t) t'"
   and "accessStore l (Stack st) = Some (KMemptr p')"
   and "st'' = updateStore l (KMemptr (ShowL_nat (Toploc (Memory st)))) (Stack st)"
   and "cps2m p (ShowL_nat (Toploc (Memory st))) x t (Storage st (Address env)) (snd (allocate(Memory
st))) = Some m"
   and "st' = st(|state.Gas := g', Memory := m, Stack := st''|)"
```

```
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 12 = assms(4-10)
  have nonChangedStack:"∀loc. loc ≠ l ⟶ accessStore loc (Stack st) = accessStore loc (Stack st')"
using 12 unfolding accessStore_def updateStore_def by auto
  have accessLStack:"accessStore l (Stack st') = Some (KMemptr (ShowL_nat (Toploc (Memory st))))" using
12 unfolding accessStore_def updateStore_def by auto
  have sameStorage:"Storage st' = Storage st " using 12 by auto
  have lInDen:"(type.Memory t', Stackloc l) |∈| fmran (Denvalue env)" using lexpStackloc_imps_inDen[of
lv env cd] 12(2) by simp
  have nonLocChanged:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs
(snd (allocate (Memory st))) = accessStore locs m"
    using 12(6) unfolding cps2m_def using cps2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t
"(Storage st (Address env))" "(snd (allocate (Memory st)))" x m]
    by simp
  have a30:"∀locs. locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ TypedStoSubpref locs
      (ShowL_nat (Toploc (Memory st))) (STArray x t) ⟶ accessStore locs (snd (allocate (Memory st))) =
      accessStore locs m"
    using 12(6) unfolding cps2m_def using cps2mSingleChange_typedPref[of p "(ShowL_nat (Toploc (Memory
st)))" t "(Storage st
        (Address env))" "(snd (allocate (Memory st)))" x m] by fastforce
  have selfPoint:"∀l l'. l ≠ (ShowL_nat (Toploc (Memory st))) ∧ TypedStoSubpref l
      (ShowL_nat (Toploc (Memory st))) (STArray x t) ∧ accessStore l m = Some (MPointer l') ⟶ l'
      = l"
    using 12(6) unfolding cps2m_def using cps2mSelfPointers[of p "(ShowL_nat (Toploc (Memory st)))" t
"(Storage st (Address env))" " (snd (allocate (Memory st)))" x m ] by blast
  have mInStd:"m = Memory st'" using 12 by simp

  obtain memArr where memArrDef: "t' = MTArray x memArr" using 12(3) cps2mTypeCompatible.simps
    by (metis mtypes.exhaust)

  have p'MCon:"MCon t' (Memory st) p'" using 2 12(4) unfolding TypeSafe_def typeCompat_def using
lInDen by (auto split:denvalue.splits option.splits stackvalue.splits type.splits)

  have MConsrc:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st⦇Gas := state.Gas st -
costs (ASSIGN lv ex) env cd
      st⦈)) (Address env)) ∧ (∃xx. KStoptr p = KStoptr xx)"
    using 2(1) 12(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st⦈)"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KStoptr p" "type.Storage (STArray x t)" g]

    using 2(3) unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )
  have "(∃p. accessStore p' (Memory st) = Some (MPointer p)) ∨ accessStore p' (Memory st) = None" us-
ing p'MCon 12(3,4,5)
    using cps2mTypeCompatible.elims(2) by fastforce

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have MCondest:"MCon (MTArray x memArr) m (ShowL_nat (Toploc (Memory st)))" using 12(6,3) unfold-
ing cps2m_def
    using cps2m[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))" " (snd (allocate
(Memory st)))"
        x m memArr] MConsrc extractValueType.simps(2) memArrDef by simp
  have selfPoint2:"∀l l'. LSubPrefL2 l (ShowL_nat (Toploc (Memory st))) ∧ accessStore l (Memory st') =
      Some (MPointer l') ⟶ l' = l" using selfPoint limitSt1 limitSt
      "12"(6) a30 cpm2m_iter_dest_unchanged allocateSameAccess cpm2m_def eq_imp_le hash_inequality
not_Some_eq
```

```
    by (metis cps2mAccessPrePost cps2m_def mInStd)

  have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                          ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto

  have noC:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (Memory st)
= accessStore locs (Memory st')"
    using allocateSameAccess nonLocChanged mInStd by metis
  have cc:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) t' ⟶ accessStore locs
(Memory st) = accessStore locs (Memory st')"
    using a30 mInStd compatible_TypedStoSubpref_imps_TypedMemSubPref[OF 12(3)] allocateSameAccess
    by (metis (no_types, lifting) "12"(6) cps2mSingleChange cps2m_def)

  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 12 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)"
      using compPointers_stack_update[OF a0 nonChangedStack accessLStack] .
  next
    show "safeContract (Accounts st') (Storage st')" using sameStorage using 12 2(1) unfolding
TypeSafe_def safeContract_def by auto
  next
    show "balanceTypes (Accounts st')" using 12 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
(Stack st) = None) ∧
            (∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st). LSubPrefL2
loc (ShowL_nat tloc)))) "
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 12 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
    proof intros

      fix tloc loc
      assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
      then show "accessStore loc (Stack st') = None" using *
        by (metis "12"(4) nonChangedStack not_None_eq)
    next
      fix loc y
      assume h1:" accessStore loc (Stack st') = Some y"
      then show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
        by (metis "12"(4) nonChangedStack)
    qed
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    have a10:"Toploc (snd (allocate (Memory st))) = Toploc m" using cps2mTopLocSame 12(6) mInStd un-
folding cps2m_def by blast
    have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp

    have a30: "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (Memory
```

```
st) = accessStore locs m" using nonLocChanged
      by (metis allocateSameAccess)
    have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
    show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
    proof intros
      fix tloc loc
      assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then have b20:"Toploc (Memory st) ≤ tloc" using a10 tloc mInStd by simp
      have "¬LSubPrefL2 loc p'"
      proof(rule ccontr)
        assume c10:"¬ ¬ LSubPrefL2 loc p'"
        then have c20: "LSubPrefL2 loc p'" by simp
        then have c30:"LSubPrefL2 p' (ShowL_nat tloc)" using b10 MemLSubPrefTransitive by auto
        then show False
        proof -
          have "MCon t' (Memory st) p'" using 12(4) 2(1) unfolding TypeSafe_def typeCompat_def using
lInDen by fastforce
          then obtain x i where c40: "accessStore p' (Memory st) = Some x ∨ accessStore (hash p'
(ShowL_nat i)) (Memory st) = Some x" using MCon_imps_Some by blast
          then show ?thesis
          proof(cases "accessStore p' (Memory st) = Some x")
            case True
            then show ?thesis using lessThanSome_imps_Locs[OF a15 True c30] b20 by simp
          next
            case False
            then have "accessStore (hash p' (ShowL_nat i)) (Memory st) = Some x" using c40 by simp
            then show ?thesis using lessThanSome_imps_Locs_hash c30 a15 b20 by fastforce
          qed
        qed
      qed
      then show "accessStore loc (Memory st') = None " using b10 b20 nonLocChanged a10 mInStd a15 a30
        by (metis (no_types, lifting) LSubPrefL2_def MemLSubPrefTransitive antisym_conv2
hash_inequality hash_suffixes_associative hashesIntSame lessThanTopLocs_def order_less_le_trans tloc)

    next
      fix loc y
      assume "accessStore loc (Memory st') = Some y "
      then show "∃tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
        by (metis a10 a15 a30 lessThanTopLocs_def order_less_trans mInStd tloc)
    qed
  next
    show ns:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"
      show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using accessLStack nonChangedStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) accessLStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.distinct(3) stackvalue.simps(17) option.inject)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) accessLStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
```

```
stackvalue.distinct(7) stackvalue.simps(18) option.inject)
        next
          case (KMemptr x3)
          then have "∃struct. t'' = type.Memory struct"
          proof(cases "x1 = l")
            case True
            have "accessStore l (Stack st) = Some (KMemptr p')" using 12(4) by simp
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) KMemptr True unfolding
TypeSafe_def typeCompat_def by (cases t'';force+)
          next
            case False
            then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')" using accessLStack
nonChangedStack by simp
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) KMemptr unfolding TypeSafe_def
typeCompat_def by (cases t'';force+)
          qed
          then obtain struct where structdef:"t'' = type.Memory struct" by blast


          have "MCon struct (Memory st') x3"
          proof(cases "x1 = l")
            case True
            then have x3Def:"x3 = (ShowL_nat (Toploc (Memory st)))" using 12 KMemptr adef unfolding
accessStore_def updateStore_def by auto
            have "(type.Memory struct, Stackloc x1) |∈| fmran (Denvalue env)"
              using Stackloc inDen structdef by blast
            then have "t' = struct" using "12"(4) True lInDen 2(1)
              unfolding TypeSafe_def unique_locations_def by auto

            then show ?thesis using memArrDef MCondest 12 Stackloc True inDen lInDen mInStd structdef
x3Def unfolding accessStore_def updateStore_def by metis
          next
            case False
            then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')" using accessLStack
nonChangedStack by simp
            then have "accessStore x1 (Stack st) = Some (KMemptr x3)" using adef KMemptr by simp
            then have mgh:"MCon struct (Memory st) x3" using inDen KMemptr Stackloc adef structdef
2(1) unfolding TypeSafe_def typeCompat_def by force

            then show ?thesis using originalMConStillMCon[OF mgh limitSt noC] cc mInStd
allocateSameAccess by metis
          qed
          then show ?thesis using structdef KMemptr Stackloc adef inDen Stackloc 2(1) unfolding
TypeSafe_def by simp

        next
          case (KStoptr x4)
          then show ?thesis using Stackloc adef sameStorage inDen Stackloc KStoptr 2(1) accessLStack
nonChangedStack unfolding TypeSafe_def typeCompat_def
            apply(cases t'')
               apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(20) type.distinct(3) type.simps(17) lInDen old.prod.inject snd_eqD uniqueLocs)
              apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(20) type.distinct(7) type.simps(18) lInDen old.prod.inject snd_eqD uniqueLocs)

             apply (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.distinct(11) stackvalue.simps(20) type.simps(19) option.inject)

            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(20)
type.distinct(11) type.simps(20) lInDen prod.inject snd_conv uniqueLocs)
        qed

        next
        case (Storeloc x2)
```

```
      then show ?thesis using sameStorage inDen 2(1) unfolding TypeSafe_def typeCompat_def by
(cases t''; force)
    qed
  qed


  next
  have "Accounts st'= Accounts st" using 12 by auto
  then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) 12 unfolding
fullyInitialised_def updateStore_def accessStore_def by auto
  next

  show "denvalueTypeCorrectness env (Stack st') (Memory st')"
    unfolding denvalueTypeCorrectness_def
  proof(intros)
    fix t2 l2 ptr_loc sub_loc
    assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env) ∧ accessStore l2 (Stack st') =
Some (KMemptr ptr_loc)"

    show "case t2 of
     MTArray len arr ⇒
       (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
    proof(cases "ptr_loc = (ShowL_nat (Toploc (Memory st)))")
      case False
      then have sameACC:"accessStore l2 (Stack st') = accessStore l2 (Stack st)"
        using nonChangedStack * 12(5) unfolding accessStore_def updateStore_def
        by (metis stackvalue.inject(3) accessLStack accessStore_def option.inject)
      then have mcOld:"MCon t2 (Memory st) ptr_loc" using * by (metis 2(1) sameMemTSafe)
      then have old:"(case t2 of
                      MTArray len arr ⇒
                        (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
                      | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using sameACC 2(1) * TypeSafe_def denvalueTypeCorrectness_def by fastforce

      have inDenvalue:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env)" using * by simp
      have lims:"¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 (ShowL_nat
(Toploc (Memory st))) ptr_loc"
        using typeSafe_noDenElementOverToploc_mem[OF 2(1) inDenvalue] * sameACC False
        using LSubPrefL2_def MemLSubPrefTransitive by metis

      then show ?thesis
      proof(cases t2)
        case (MTArray x11 x12)
        have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12)"
using old MTArray by simp

        have nonLocChanged_TypedSafe:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                    ⟶ accessTypeStore loc (Memory st') = accessTypeStore loc
(Memory st)"
          using lims mInStd 12(6) unfolding cps2m_def
          by (metis allocateTypeSameAccess cps2mSingleChange_Typed cps2m_def)
        have nonLocChanged_TypedSafe2:"∀ loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                    ⟶ accessStore loc (Memory st') = accessStore loc (Memory st)"
          using lims mInStd 12(6) unfolding cps2m_def
          by (metis allocateSameAccess cps2mSingleChange cps2m_def)
        have "∀ i. ¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) (ShowL_nat (Toploc (Memory st)))"
          using lims Mutual_NonSub_SpecificNonSub by blast
        then have conc0:"∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some
x12"
          using old' lims nonLocChanged_TypedSafe by auto
        then show ?thesis using MTArray old lims nonLocChanged_TypedSafe2
          using nonLocChanged_TypedSafe by auto
```

```
          next
          case (MTValue x2)
          have "¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc (Memory st)))" using lims by simp
          then have "accessTypeStore ptr_loc (Memory st') = accessTypeStore ptr_loc (Memory st)"
              by (metis "12"(6) allocateTypeSameAccess cps2mSingleChange_Typed cps2m_def mInStd)
          then show ?thesis using MTValue old by simp
        qed
        next
        case True
        then have newLocation:"l2 = l ∧ t2 = (MTArray x memArr)"
            using lInDen accessLStack * memArrDef 2(1) "12"(3,5,7) LSubPrefL2_def nonChangedStack
snd_eqD typeSafeUnique typeSafe_noDenElementOverToploc_mem
              unique_locations_def
            by (metis Pair_inject type.inject(3))

        then have l2IsL:"l2 = l" using newLocation by simp
        then have conc0:"ptr_loc = (ShowL_nat (Toploc (Memory st)))" using * 12(5,7) by auto
        have conc1:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) (Memory
st') = Some memArr"
            using 12(3,6) memArrDef unfolding cps2m_def
            using cps2m_TypeCompChangeIndexs[of p "(ShowL_nat (Toploc (Memory st)))" t _ "(snd (allocate
(Memory st)))" x m]
              mInStd by blast
        show ?thesis using conc1 conc0 newLocation by simp
      qed
    qed
    next
    have old:"subPrefixStructuralConsistency (Memory st)" using 2(1) unfolding TypeSafe_def by simp
    have noC:"∀ locs.
   ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs = (ShowL_nat (Toploc (Memory st))) ⟶
   accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        by (metis (lifting) "12"(6) allocateTypeSameAccess cps2mSingleChange_Typed cps2m_def mInStd)
    have some:"SomeValSomeTyp (Memory st)" using 2(1) unfolding TypeSafe_def by simp
    have less:"lessThanTopLocs (Memory st)" using 2(1) unfolding TypeSafe_def by simp
    have n:"accessStore (ShowL_nat (Toploc (Memory st))) (Memory st) = None" using limitSt
        using ‹accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None›
allocateSame by presburger

    have cc:"∀ locs.
     ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr) ∨ locs = (ShowL_nat
(Toploc (Memory st))) ⟶
     accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')" using memArrDef
        by (metis (lifting) "12"(3,6) allocateTypeSameAccess compatible_TypedStoSubpref_imps_TypedMemSubPref_neg
            cps2mSingleChange_typedPref_typed cps2m_def mInStd noC)
    have tps:"∀ destl'.
   TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x memArr) ⟶
   (∃ stt. CompMemType (Memory st') x memArr stt (ShowL_nat (Toploc (Memory st))) destl' ∧
          (case stt of
           MTArray parent_len parent_arr ⇒
             ∀ i<parent_len. accessTypeStore (hash destl' (ShowL_nat i)) (Memory st') = Some parent_arr
           | MTValue pval ⇒ accessTypeStore destl' (Memory st') = Some (MTValue pval)))"
        using 12(3,6) unfolding cps2m_def
        using cps2m_TypeCompChange[of p "(ShowL_nat (Toploc (Memory st)))" t _ "(snd (allocate (Memory
st)))" x m memArr]
          mInStd memArrDef by blast
    have mc:"MCon (MTArray x memArr) (Memory st') (ShowL_nat (Toploc (Memory st)))"
        using MCondest mInStd by blast
    have conc1:"∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) (Memory st')
= Some memArr"
        using 12(3,6) memArrDef unfolding cps2m_def
        using cps2m_TypeCompChangeIndexs[of p "(ShowL_nat (Toploc (Memory st)))" t _ "(snd (allocate
(Memory st)))" x m]
          mInStd by blast
```

```
    have noC2:"∀locs.
     ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs = (ShowL_nat (Toploc (Memory st))) ⟶
     accessStore locs ((Memory st)) = accessStore locs (Memory st')"
        using "12"(6) unfolding cps2m_def
        using cps2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))"
"(snd (allocate (Memory st)))" x m]
          mInStd allocateSameAccess by metis
    have nocc2:"∀locs.
     ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr) ⟶
     accessStore locs (Memory st) = accessStore locs (Memory st')"
        using "12"(6) unfolding cps2m_def
        using cps2mSingleChange_typedPref[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address
env))" "(snd (allocate (Memory st)))" x m]
          mInStd allocateSameAccess compatible_TypedStoSubpref_imps_TypedMemSubPref[OF 12(3)]
        by (metis memArrDef noC2)

    show "subPrefixStructuralConsistency (Memory st')"
        using cpm2m_subPrefixPersist[OF old n some noC less cc tps mc _ conc1 noC2 nocc2]
        using selfPoint2 typedPrefix_imp_SubPref by blast
    next
    have old:"(∀locs. (∃t. accessStore locs (Memory st) = Some t) = (∃tt. accessTypeStore locs (Memory
st) = Some tt))"
        using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    have somesome:"∀destl'. TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)
          ⟶ (∃t. accessStore destl' m = Some t) = (∃tt. accessTypeStore destl' m = Some tt)"
        using 12(6,3) unfolding cps2m_def
        using cps2m_TypeCompChange_somesome[of p "(ShowL_nat (Toploc (Memory st)))" t _ "(snd (allocate
(Memory st)))" x m]
          compatible_TypedStoSubpref_imps_TypedMemSubPref memArrDef by blast
    have a30T:"∀locs.
     locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ TypedStoSubpref locs (ShowL_nat (Toploc (Memory st)))
(STArray x t) ⟶
     accessTypeStore locs (snd (allocate (Memory st))) = accessTypeStore locs (Memory st')"
        using 12(6,3) unfolding cps2m_def using cps2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t _ " (snd (allocate (Memory st)))" x m ]
        using mInStd by blast
    then have a30T:"∀locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)
⟶
     accessTypeStore locs (snd (allocate (Memory st))) = accessTypeStore locs (Memory st')"
        using mInStd allocateTypeSameAccess memArrDef compatible_TypedStoSubpref_imps_TypedMemSubPref[OF
12(3)]
        by (metis "12"(6) cps2mSingleChange_Typed cps2m_def)

    show "SomeValSomeTyp (Memory st')"unfolding SomeValSomeTyp_def
    proof intros
      fix locs
      show "(∃t. accessStore locs (Memory st') = Some t) = (∃tt. accessTypeStore locs (Memory st') =
Some tt) "
      proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)")
        case True
        then have "(∃t. accessStore locs m = Some t) = (∃tt. accessTypeStore locs m = Some tt)"
            using somesome by simp
        then show ?thesis using mInStd by simp
        next
        case False
        then have "accessStore locs (Memory st) = accessStore locs (Memory st')"
            using a30 mInStd allocateSameAccess
            by (metis (mono_tags, lifting) "12"(3,6) compatible_TypedStoSubpref_imps_TypedMemSubPref_neg
cps2mSingleChange cps2m_def
                memArrDef)
        moreover have "accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
            using a30T False
            by (metis allocateTypeSameAccess)
        ultimately show ?thesis using old by metis
```

```
      qed
    qed
    next
    fix locs tt
    have a30T:"∀ locs.
     locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ TypedStoSubpref locs (ShowL_nat (Toploc (Memory st)))
(STArray x t) ⟶
     accessTypeStore locs (snd (allocate (Memory st))) = accessTypeStore locs (Memory st')"
        using 12(6,3) unfolding cps2m_def using cps2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t _ " (snd (allocate (Memory st)))" x m ]
        using mInStd by blast

    assume a1:"accessTypeStore locs (Memory st) = Some tt"
    then have some:"∃ v. accessStore locs (Memory st) = Some v"
        using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by auto
    show "accessTypeStore locs (Memory st') = Some tt "
    proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)")
      case True
      then show ?thesis using mInStd a1 some
          by (metis le_refl limitSt option.distinct(1) typedPrefix_imp_SubPref)
      next
      case False

      then have "accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
        using a30T ‹accessStore ⌊Toploc (state.Memory st)⌋ (snd (allocate (state.Memory st))) = None›
          assms(6) memArrDef some
        by (metis allocateSameAccess allocateTypeSameAccess compatible_TypedStoSubpref_imps_TypedMemSubPref
           option.distinct(1))
      then show ?thesis using a1 by auto
    qed
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
        by (metis allocateSameAccess dual_order.eq_iff limitSt mInStd nonLocChanged not_None_eq)
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
      using limitSt noC by (metis dual_order.refl option.distinct(1))
    next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using allocateSameAccess limitSt mInStd nonLocChanged
        by (metis LSubPrefL2_def MemLSubPrefTransitive hash_flatten_right hash_inequality hashesIntSame
nat_neq_iff)
    next
    have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
    moreover have topLocEq:"Toploc (snd (allocate (Memory st))) = Toploc (Memory st')"
        using cps2mTopLocSame 12(6) cps2m_def mInStd MCondest
        by (metis (no_types, lifting))
    ultimately show "Toploc (Memory st) ≤ Toploc (Memory st')" by simp
  qed
qed
```

Assignment case 13: a storage array assigned to a storage pointer on the stack: the stack now points to the same storage array.

```
lemma TypeSafe_Assign_13:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st⦇state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KStoptr p, type.Storage (STArray x t)), g)"
    and "lexp lv env cd (st⦇state.Gas := g⦈) g = Normal((LStackloc l, type.Storage t'),g')"
```

```
    and "t' = STArray x t"
    and "st' = st(|state.Gas := g', Stack := updateStore l (KStoptr p) (Stack st)|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 13 = assms(4-7)
  then have stackChanged:"∀l'. l' ≠ l ⟶ accessStore l' (Stack st) = accessStore l' (Stack st')"
unfolding updateStore_def accessStore_def by simp
  have t'InDen:"(type.Storage t', Stackloc l) |∈| fmran (Denvalue env)" using
lexpStackloc_imps_inDen[OF 13(2)] by blast
  then have lDen:"∀t'. (t', Stackloc l) |∈| fmran (Denvalue env) ⟶ t'= (type.Storage (STArray x t))"
using 13 2(1) unfolding TypeSafe_def unique_locations_def by auto

  have pOrigin:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st(|Gas := state.Gas st -
costs (ASSIGN lv ex) env cd st|)) (Address env)) ∧
      (∃xx. KStoptr p = KStoptr xx) ∧
      (∃stloc tp'' .
          (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue env) ∧
          (tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc ∨ extractValueType (KStoptr p) ≠ stloc ∧
CompStoType tp'' (STArray x t) stloc (extractValueType (KStoptr p))))"
    using 2(1) 13(1) 2(3) using
      exprTypeconInduct(3)[of ex env cd "(st(|Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|))"
"(state.Gas st - costs
          (ASSIGN lv ex) env cd st)" "KStoptr p" "type.Storage (STArray x t)" g]
    by (auto split:type.splits if_splits )
  obtain pParent pParentT where
    pOrigin:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st(|Gas := state.Gas st - costs
(ASSIGN lv ex) env cd st|)) (Address env))
        ∧(type.Storage pParentT, Storeloc pParent) |∈| fmran (Denvalue env) ∧
          (pParentT = (STArray x t) ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p) ≠
pParent ∧
              CompStoType pParentT (STArray x t) pParent (extractValueType (KStoptr p)))"
    using pOrigin by auto
  have a0: "compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  then have pParentRelations:"(∀tp2 l2 l2' stl2.
      (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧

      (l2 = Stackloc l2' ∧ accessStore l2' (Stack (st)) = Some (KStoptr stl2) ∨ l2 = Storeloc stl2) ⟶
      (if TypedStoSubpref stl2 pParent pParentT then CompStoType pParentT tp2 pParent stl2 else if
TypedStoSubpref pParent stl2 tp2 then CompStoType tp2 pParentT stl2 pParent else True))"
    using compPointers_def pOrigin by blast
  have pSCon:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st) (Address env))" using
pOrigin by force

  have pRelations:"(∀tp2 l2 l2' stl2 .
  (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
  ((l2 = Stackloc l2' ∧ accessStore l2' (Stack st) = Some(KStoptr stl2)) ∨ (l2 = Storeloc stl2))
   ⟶
  (if TypedStoSubpref p stl2 tp2 then CompStoType tp2 (STArray x t) stl2 p
   else if TypedStoSubpref stl2 p (STArray x t) then CompStoType (STArray x t) tp2 p stl2
   else True))"
  proof intros
    fix tp2 l2 l2' stl2
    assume in1:"(type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧ (l2 = Stackloc l2' ∧ accessStore l2'
(Stack st) = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"

    show "if TypedStoSubpref p stl2 tp2 then CompStoType tp2 (STArray x t) stl2 p else if
TypedStoSubpref stl2 p (STArray x t) then CompStoType (STArray x t) tp2 p stl2 else True"
    proof(cases "l2 = Stackloc l2'")
      case StL2:True
      then have mcStl2:"SCon tp2 stl2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using in1 by fastforce
```

```
      then have "SCon pParentT pParent (Storage st (Address env))" using pOrigin using 2(1) unfolding
TypeSafe_def typeCompat_def by force
      have prnt:" pParentT = STArray x t ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p)
≠ pParent ∧
          CompStoType pParentT (STArray x t) pParent (extractValueType (KStoptr p))"
        using pOrigin by simp
      then have comp:"(if TypedStoSubpref stl2 pParent pParentT then CompStoType pParentT tp2 pParent
stl2
                    else if TypedStoSubpref pParent stl2 tp2 then CompStoType tp2 pParentT stl2 pParent
else True)"
        using pParentRelations in1 StL2 prnt by force
      then show ?thesis
      proof(cases "TypedStoSubpref p stl2 tp2")
        case pToStl2:True

        then show ?thesis
        proof(cases "pParentT = (STArray x t)")
          case True
          then have cc1:"KStoptr p = KStoptr pParent " using prnt
            using comp_stotype_same_type_same_loc by blast
          then have cc2:"TypedStoSubpref pParent stl2 tp2" using pToStl2 by auto
          then have "CompStoType tp2 (STArray x t) stl2 p"
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case t2:True
            then show ?thesis using comp
              by (metis CompStoType_sameLoc_sameType stackvalue.inject(4) True cc1 cc2
typedStoSub_imps_negInv)
          next
            case False
            then have "CompStoType tp2 pParentT stl2 pParent" using comp
              by (simp add: cc2)
            then show ?thesis
              using True cc1 by auto
          qed
          then show ?thesis by (simp add: pToStl2)
        next
          case False
          then have cmp:" CompStoType pParentT (STArray x t) pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref by
simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            have "CompStoType tp2 (STArray x t) stl2 p " using pToStl2 prntStl2 CompStoType_sharedSub
cmp by simp
            then show ?thesis using pToStl2 by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STArray x t) stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
              then show ?thesis using pToStl2 by simp
            next
              case False
              then have "¬TypedStoSubpref p stl2 tp2" using f1 cmp tsp NotRelatedPrnt_imps_notRelatedChild[of
stl2 pParent pParentT tp2 p] by simp
              then show ?thesis using f1 pToStl2 cmp by simp
            qed
          qed
        qed
```

```
      next
        case f1:False
        then show ?thesis
        proof(cases " TypedStoSubpref stl2 p (STArray x t)")
          case stl2ToP:True
          have " CompStoType (STArray x t) tp2 p stl2"
          proof(cases "pParentT = (STArray x t)")
            case True
            then have cc1:"KStoptr p = KStoptr pParent " using prnt
              using comp_stotype_same_type_same_loc by blast
            then have cc2:"TypedStoSubpref stl2 pParent (STArray x t)" using stl2ToP by auto
            then have " CompStoType (STArray x t) tp2 p stl2"
            proof(cases "TypedStoSubpref stl2 pParent pParentT")
              case t2:True
              then have "CompStoType pParentT tp2 pParent stl2" using comp by simp
              then have "CompStoType (STArray x t) tp2 p stl2" using cc1 True by blast
              then show ?thesis by blast
            next
              case False
              then show ?thesis
              proof(cases "TypedStoSubpref pParent stl2 tp2")
                case True
                then have "CompStoType tp2 pParentT stl2 pParent" using comp False by simp
                then show ?thesis using True cc1 f1 by auto
              next
                case f2:False
                then show ?thesis using False True cc2 by blast
              qed
            qed
            then show ?thesis by blast
          next
            case False
            then have cmp:" CompStoType pParentT (STArray x t) pParent p" using prnt by simp
            then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref
by simp
            then show ?thesis
            proof(cases "TypedStoSubpref stl2 pParent pParentT")
              case True
              then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
              then show ?thesis using CompStoType_sharedSub[OF prntStl2 stl2ToP cmp] by simp
            next
              case f1:False
              then show ?thesis
              proof(cases "TypedStoSubpref pParent stl2 tp2")
                case True
                then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
                then have "CompStoType tp2 (STArray x t) stl2 p" using cmp CompStoType_trns[of tp2 ]
by blast
                then show ?thesis
                  using CompStoType_sameLocNdTyp CompStoType_sharedSub stl2ToP by blast
              next
                case False
                then show ?thesis
                  using NotReachablePrnt_imps_notReachableChild cmp f1 stl2ToP by blast
              qed
            qed
          qed
          then show ?thesis using f1 by (simp )
        next
          case False
          then show ?thesis
            using f1 by auto
        qed
      qed
```

```
    next
      case f3:False
      then have StL2:"l2 = Storeloc stl2"
        using in1 by auto

      then have "SCon pParentT pParent (Storage st (Address env))" using pOrigin using 2(1) unfolding
TypeSafe_def typeCompat_def by force
      have prnt:" pParentT = STArray x t ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p)
≠ pParent ∧
         CompStoType pParentT (STArray x t) pParent (extractValueType (KStoptr p))"
        using pOrigin by simp
      then have comp:"(if TypedStoSubpref stl2 pParent pParentT then CompStoType pParentT tp2 pParent
stl2
                  else if TypedStoSubpref pParent stl2 tp2 then CompStoType tp2 pParentT stl2 pParent
else True)"
        using pParentRelations in1 StL2 prnt by force
      then show ?thesis
      proof(cases "TypedStoSubpref p stl2 tp2")
        case pToStl2:True

        then show ?thesis
        proof(cases "pParentT = (STArray x t)")
          case True
          then have cc1:"KStoptr p = KStoptr pParent " using prnt
            using comp_stotype_same_type_same_loc by blast
          then have cc2:"TypedStoSubpref pParent stl2 tp2" using pToStl2 by auto
          then have "CompStoType tp2 (STArray x t) stl2 p"
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case t2:True
            then show ?thesis using comp
              by (metis CompStoType_sameLoc_sameType stackvalue.inject(4) True cc1 cc2
typedStoSub_imps_negInv)
          next
            case False
            then have "CompStoType tp2 pParentT stl2 pParent" using comp
              by (simp add: cc2)
            then show ?thesis
              using True cc1 by auto
          qed
          then show ?thesis by (simp add: pToStl2)
        next
          case False
          then have cmp:" CompStoType pParentT (STArray x t) pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref by
simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            have "CompStoType tp2 (STArray x t) stl2 p " using pToStl2 prntStl2 CompStoType_sharedSub
cmp by simp
            then show ?thesis using pToStl2 by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STArray x t) stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
              then show ?thesis using pToStl2 by simp
            next
              case False
```

```
            then have "¬TypedStoSubpref p stl2 tp2" using f1 cmp tsp NotRelatedPrnt_imps_notRelatedChild[of
stl2 pParent pParentT tp2 p] by simp
            then show ?thesis using f1 pToStl2 cmp by simp
          qed
        qed
      qed
    next
      case f1:False
      then show ?thesis
      proof(cases " TypedStoSubpref stl2 p (STArray x t)")
        case stl2ToP:True
        have " CompStoType (STArray x t) tp2 p stl2"
        proof(cases "pParentT = (STArray x t)")
          case True
          then have cc1:"KStoptr p = KStoptr pParent " using prnt
            using comp_stotype_same_type_same_loc by blast
          then have cc2:"TypedStoSubpref stl2 pParent (STArray x t)" using stl2ToP by auto
          then have " CompStoType (STArray x t) tp2 p stl2"
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case t2:True
            then have "CompStoType pParentT tp2 pParent stl2" using comp by simp
            then have "CompStoType (STArray x t) tp2 p stl2" using cc1 True by blast
            then show ?thesis by blast
          next
            case False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp False by simp
              then show ?thesis using True cc1 f1 by auto
            next
              case f2:False
              then show ?thesis using False True cc2 by blast
            qed
          qed
          then show ?thesis by blast
        next
          case False
          then have cmp:" CompStoType pParentT (STArray x t) pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref
by simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            then show ?thesis using CompStoType_sharedSub[OF prntStl2 stl2ToP cmp] by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STArray x t) stl2 p" using cmp CompStoType_trns[of tp2 ]
by blast
              then show ?thesis
                using CompStoType_sameLocNdTyp CompStoType_sharedSub stl2ToP by blast
            next
              case False
              then show ?thesis
                using NotReachablePrnt_imps_notReachableChild cmp f1 stl2ToP by blast
            qed
          qed
        qed
        then show ?thesis using f1 by (simp )
```

```
        next
          case False
          then show ?thesis using f1 by auto
        qed
      qed
    qed
  qed

  have storageSame:"(Storage st' (Address env)) = (Storage st (Address env))" using 13 by simp
  have memorySame:"Memory st = Memory st'" using 13 by simp
  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof (intros)
    show tcN:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t l'"
      proof (split denvalue.split, intros)
        fix loc assume a20:"l' = Stackloc loc"
        show "case accessStore loc (Stack st') of None ⇒ False
        | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
        | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
        | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
        | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
        proof (cases "loc = l")
          case False
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then have "accessStore loc (Stack st) = None" using 13(4) False by (simp
add:stackSingleUpdate)
            then show False using a30 a20 a10 assms TypeSafe_def False 2(1)
              by (metis option.distinct(1) typeSafeLocExists)
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have a40:"accessStore loc (Stack st) = Some x2" using 13(4) False by (simp
add:stackSingleUpdate)
            then have a50:"(Memory st) = (Memory st')" using 13(4) by simp
            then have a60:"(Storage st) = (Storage st')" using 13(4) by simp
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False )
                | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
                | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _
⇒ False)
                | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
              using a10 a20 a30 a40 a50 a60 2(1) unfolding TypeSafe_def typeCompat_def by (cases x2;
cases t; force)
          qed
        next
          case sameLoc:True
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then show False using a20 a10 assms(1) sameLoc 13(4) notNoneUpdate[of st' g' loc "(KMemptr
p)"] by simp
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have x2IsP:"x2 = KStoptr p" using 13 sameLoc by auto
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False )
            | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
            | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
```

```
          | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
          proof(cases "x2")
            case (KValue x1)

            then show ?thesis using x2IsP by auto
          next
            case (KCDptr x2)
            then show ?thesis using x2IsP by (simp add:sameLoc)
          next
            case (KMemptr x3)
            then show ?thesis using x2IsP by (simp add:sameLoc)
          next
            case (KStoptr x4)
            then show ?thesis using x2IsP a10 a20 lDen storageSame sameLoc
              using pSCon by force
          qed
        qed
      qed
      next
        fix x2 assume a20:"l' = Storeloc x2"
        then have "(Storage st' (Address env)) = (Storage st (Address env))" using 13(4) by simp
        then show "case t of type.Storage typ ⇒ SCon typ x2 (Storage st' (Address env)) | _ ⇒ False"
using a10 a20 TypeSafe_def 2(1) pSCon a10 a20 lDen storageSame
        proof -
          have "⋀t d. (t, d) |∉| fmran (Denvalue env) ∨
                (case d of Stackloc l ⇒
                (case accessStore l (Stack st) of None ⇒ False
                | Some (KValue l) ⇒ (case t of Value t ⇒ typeCon t l | _ ⇒ False)
                | Some (KCDptr l) ⇒ (case t of Calldata m ⇒ MCon m cd l | _ ⇒ False)
                | Some (KMemptr l) ⇒ (case t of type.Memory m ⇒ MCon m (Memory st) l | _ ⇒ False)
                | Some (KStoptr l) ⇒ (case t of type.Storage s ⇒ SCon s l (Storage st (Address env)) |
_ ⇒ False))
                | Storeloc l ⇒ (case t of type.Storage s ⇒ SCon s l (Storage st (Address env)) | _ ⇒
False))"
            using 2(1) TypeSafe_def typeCompat_def by auto
          then show ?thesis
            by (metis (no_types) denvalue.simps(6) a10 a20 storageSame)
        qed
      qed
    qed
    have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st) (Memory st') (Storage st' (Address env)) cd t l' "
      proof(cases l')
        case (Stackloc x1)
        then show ?thesis
        proof(cases "x1 = l")
          case sameLoc:True
          then show ?thesis
          proof (cases "accessStore x1 (Stack st)")
            case None
            then show ?thesis using Stackloc a10 assms(1) sameLoc 13(4)
              by (metis 2(1) option.distinct(1) typeSafeAllStacklocsExist)
          next
            case (Some a)
            then show ?thesis
            proof(cases "a")
              case (KValue x1)
              then show ?thesis using a10 Stackloc Some 2(1) unfolding TypeSafe_def typeCompat_def
                by fastforce
            next
```

```
            case (KCDptr x2)
            then show ?thesis using a10 Stackloc Some 2(1) unfolding TypeSafe_def typeCompat_def
              by fastforce
          next
            case (KMemptr x3)
            then show ?thesis using a10 Stackloc Some 2(1) unfolding TypeSafe_def typeCompat_def
              by (metis (lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(19)
type.simps(20) sameLoc t'InDen)
          next
            case (KStoptr x4)
            then show ?thesis using a10 Stackloc Some 2(1) unfolding TypeSafe_def typeCompat_def
using lDen storageSame sameLoc
              using pSCon by force
          qed
        qed
      next
        case False
        then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')"
          using stackChanged by auto
        then show ?thesis using tcN Stackloc a10 unfolding typeCompat_def by fastforce
      qed
    next
      case (Storeloc x2)
      then show ?thesis using a10 tcN unfolding typeCompat_def by fastforce
    qed
  qed
next
  show "unique_locations (Denvalue env)" using 2(1) typeSafeUnique by auto
next
  have "(Accounts st) = Accounts(st')" using 13 by simp
  then show "balanceTypes (Accounts st')" using balanceTypes_def balanceTypes_def 2(1)
typeSafeAccounts by simp
next
  have a0:"compPointers (Stack st)(Denvalue env)" using 2(1) storageSame unfolding TypeSafe_def by
simp
  show " compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
  proof(intros)
    fix tp1 tp2 l1 l22 l1' l2' stl1 stl2
    assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
 (type.Storage tp2, l22) |∈| fmran (Denvalue env) ∧
 (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
 (l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l22 = Storeloc stl2)"
    have a2:"(∀x y. x |∈| fmran (Denvalue env) ∧ y |∈| fmran (Denvalue env) ∧ snd x = snd y ⟶ x =
y)"
      using 2(1) unfolding TypeSafe_def unique_locations_def by simp

    then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
    proof(cases "l1 = Storeloc stl1")
      case t1:True
      then show ?thesis
      proof(cases "l22 = Storeloc stl2")
        case True
        then show ?thesis using t1 a0 a1 unfolding compPointers_def by blast
      next
        case False
        then have a4:"l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2)" using
a1 by simp
        then show ?thesis
        proof(cases "l2' = l")
          case True
          then have "stl2 = p" using a4 13 by simp
          then show ?thesis using pRelations a1 a0 t1 True a4 lDen by blast
        next
```

```
            case False
            then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
            then show ?thesis using a0 a1 t1 unfolding compPointers_def by metis
          qed
        qed
      next
        case f1:False
        then have a4:"l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1)" using a1
by simp
        then show ?thesis
        proof(cases "l22 = Storeloc stl2")
          case t1:True
          then show ?thesis
          proof(cases "l1' = l")
            case True
            then have "stl1 = p" using a4 13 by simp
            then show ?thesis using pRelations a1 a0 t1 True a4 lDen
              by (metis CompStoType_sameLoc_sameType type.inject(4) typedStoSub_imps_negInv)
          next
            case False
            then have " accessStore l1' (Stack st') = accessStore l1' (Stack st)" using stackChanged
a4 by simp
            then show ?thesis using a0 a1 t1 unfolding compPointers_def by metis
          qed
        next
          case False
          then have a4':"l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2)" using
a1 by simp
          then show ?thesis
          proof(cases "l1' = l")
            case True
            then have p1:"stl1 = p" using a4 13 by simp
            then have tp1c:"tp1 = (STArray x t)" using lDen a1
              using True a4 by auto
            then show ?thesis
            proof(cases "l2' = l")
              case True
              then have p2:"stl2 = p" using a4' 13 by auto
              then have tp2c:"tp2 = STArray x t" using lDen a1 True a4' by auto
              then show ?thesis using p2 p1 tp1c tp2c by auto
            next
              case False
              then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
              then show ?thesis
                by (metis CompStoType_sameLoc_sameType a1 p1 pRelations tp1c typedStoSub_imps_negInv)
            qed
          next
            case False
            then have same1:" accessStore l1' (Stack st') = accessStore l1' (Stack st)" using
stackChanged a4 by simp
            then show ?thesis
            proof(cases "l2' = l")
              case True
              then have p2:"stl2 = p" using a4' 13 by auto
              then have tp2c:"tp2 = STArray x t" using lDen a1 True a4' by auto
              then show ?thesis using same1 pRelations p2 tp2c a1
                by metis
            next
              case False
              then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
              then show ?thesis using a1 a0 same1
```

```
            using compPointers_def by auto
          qed
        qed
      qed
    qed
  qed
  next
    show "svalueTypes (Svalue env)" using svalueTypes_def typeSafeSvalue 2(1) by simp
  next
    have "(Storage st') = (Storage st)" using 13(4) by simp
    then show "safeContract (Accounts st') (Storage st')" using 2(1) 13 unfolding safeContract_def
TypeSafe_def by auto
  next
    have a10:"Toploc (Stack st') = Toploc (Stack st)" using 13(4) unfolding updateStore_def by simp
    then have a20:"∃ val. accessStore l (Stack st) = Some val" using t'InDen typeSafeLocExists 2(1)
TypeSafe_def by blast
    then have a30:"(∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st).
LSubPrefL2 loc (ShowL_nat tloc)))"
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    then have a40:"(∀ tloc loc. Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                 ∧(∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st').
LSubPrefL2 loc (ShowL_nat tloc)))"
      using a10 by simp
    show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
    proof intros
      fix tloc loc
      assume *:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then show "accessStore loc (Stack st') = None"
      proof(cases "loc = 1")
        case True
        then show ?thesis using * a10
          by (metis a20 a30 option.distinct(1))
      next
        case False
        then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 13(4) unfolding
updateStore_def accessStore_def by simp
        then show ?thesis using 2(1) a40 * a10 False a30 by simp
      qed
    next
      fix loc y
      assume *:" accessStore loc (Stack st') = Some y "
      show "∃ tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)"
      proof(cases "loc = 1")
        case True
        then show ?thesis using *a10 a20 a30 by simp
      next
        case False
        then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 13(4) unfolding
updateStore_def accessStore_def by simp
        then show ?thesis using * 2(1) a40 by simp
      qed
    qed
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def using 13 by auto
  next
    show "lessThanTopLocs (Memory st')" using 2(1) unfolding TypeSafe_def using 13 by auto
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by auto
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using 13 by simp
```

```
  next
    have accSame:"Accounts st'= Accounts st" using 13 by auto
    from 2(3) obtain c_fi ct_fi dud_fi where fi:"
         Type (Accounts st (Address env)) = Some (atype.Contract c_fi) ∧
         Contract env = c_fi ∧
         ep $$ c_fi = Some (ct_fi, dud_fi) ∧
         (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v, Storeloc
id))) ∧
         (∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
         (∀t0 l0 p0.
             (type.Storage t0, Stackloc l0) |∈| fmran (Denvalue env) ∧ accessStore l0 (Stack st) = Some
(KStoptr p0) ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t0 l'
p0))"
      unfolding fullyInitialised_def by blast
    have fiPtrs':"∀t'' l'' p''.
             (type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack st') =
Some (KStoptr p'') ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t'' l'
p'')"
    proof (intro allI impI)
      fix t'' l'' p''
      assume in1:"(type.Storage t'', Stackloc l'') |∈| fmran (Denvalue env) ∧ accessStore l'' (Stack
st') = Some (KStoptr p'')"
      then show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t''
l' p''"
      proof(cases "l'' = l")
        case True
        then have isST:"t'' = STArray x t" using in1 lDen by blast
        then have isP:"p'' = p" using 13(4) in1 True unfolding accessStore_def updateStore_def by
simp
        then have "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t'
(STArray x t) l' p"
          using pOrigin isST isP by force
        then show ?thesis by (simp add: isP isST)
      next
        case False
        then have sameAcc:"accessStore l'' (Stack st') =accessStore l'' (Stack st)"
          by (simp add: stackChanged)
        then show ?thesis using fi in1 by presburger
      qed
    qed
    show "fullyInitialised env (Accounts st') (Stack st')"
      unfolding fullyInitialised_def using fi fiPtrs' accSame by metis
  next
    have inDen:"(type.Storage t', Stackloc l) |∈| fmran (Denvalue env)" using t'InDen by simp
    have denAc:"(∀t l. (t, l) |∈| fmran (Denvalue env) ⟶
         (denvalueCon (Stack st) (Memory st) (Storage st (Address env)) cd t l))" using 2(1) unfolding
TypeSafe_def typeCompat_def by simp
    have isVal:"∃val. accessStore l (Stack st) = Some (KStoptr val)"
    proof(cases "accessStore l (Stack st)")
      case None
      then show ?thesis using inDen denAc by fastforce
    next
      case (Some a)
      then show ?thesis
      proof(cases a)
        case (KValue x1)
        then show ?thesis using inDen denAc Some by fastforce
      next
        case (KCDptr x2)
        then show ?thesis using inDen denAc Some by fastforce
      next
        case (KMemptr x3)
```

```
      then show ?thesis using inDen denAc Some by fastforce
    next
      case (KStoptr x4)
      then show ?thesis using inDen denAc Some by simp
    qed
  qed

  have memSame:"Memory st = Memory st'" using 13(4) by simp
  have cc0:"∀ l ptr. (accessStore l (Stack st) = Some (KMemptr ptr)) = (accessStore l (Stack st') =
Some (KMemptr ptr))"
  proof intros
    fix l2 ptr
    show " (accessStore l2 (Stack st) = Some (KMemptr ptr)) = (accessStore l2 (Stack st') = Some
(KMemptr ptr)) "
    proof
      assume *:"accessStore l2 (Stack st) = Some (KMemptr ptr)"
      then have "l ≠ l2" using isVal by auto
      then show "accessStore l2 (Stack st') = Some (KMemptr ptr)"
        using 13(4) * unfolding updateStore_def accessStore_def by auto
    next
      assume *:"accessStore l2 (Stack st') = Some (KMemptr ptr)"
      then have "l ≠ l2" using 13(4) by auto
      then show "accessStore l2 (Stack st) = Some (KMemptr ptr) "
        using * 13(4) unfolding updateStore_def accessStore_def by simp
    qed
  qed
next
  have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
    using 13(4) unfolding updateStore_def accessStore_def by auto
  show "denvalueTypeCorrectness env (Stack st') (Memory st')"

    unfolding denvalueTypeCorrectness_def
  proof intros
    fix t l ptr_loc sub_loc
    assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
   accessStore l (Stack st') = Some (KMemptr ptr_loc)"
    then have "(case t of
     MTArray len arr ⇒
       (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
      using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
    moreover have "Memory st = Memory st'" using 13(4) by simp
    ultimately show "case t of
   MTArray len arr ⇒
     (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
   | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
      by metis
  qed
next
  show "subPrefixStructuralConsistency (Memory st')"
    using 2(1) unfolding TypeSafe_def using 13(4) by simp
next
  show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 13(4) by simp
next
  show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
    using 2(1) unfolding TypeSafe_def using 13(4) by simp
next
  show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
    using 13(4) by simp
next
  show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
```

```
st') = Some (MValue v')"
      using 13(4) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
       (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 13(4) by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 13(4) by simp
  qed
qed
```

Assignment case 14: a storage array assigned to a storage location: the array is copied within storage.

```
lemma TypeSafe_Assign_14:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st⦇state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KStoptr p, type.Storage (STArray x t)), g)"
    and "lexp lv env cd (st⦇state.Gas := g⦈) g = Normal((LStoreloc l, type.Storage t'),g')"
    and "t' = (STArray x t)"
    and "copy p l x t (Storage st (Address env)) = Some s"
    and "st' = st⦇state.Gas := g', Storage := (Storage st) (Address env := s)⦈"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 14 = assms(4-8)
  have sameStack:"(Stack st) = (Stack st')" using 14 unfolding accessStore_def updateStore_def by auto
  have sameMemory:"Memory st' = Memory st " using 14 by auto
  have nonLocChanged:"∀ t' locs. ¬ LSubPrefL2 locs l ∨ locs = l ⟶ accessStorage t' locs (Storage st
(Address env)) = accessStorage t' locs s"
    using 14(4) unfolding copy_def using copySingleChange[of p l t "(Storage st (Address env))" x s]
    by fastforce
  have a30:"∀ locs t''. ¬ TypedStoSubpref locs l (STArray x t) ⟶ accessStorage t'' locs (Storage st
(Address env)) = accessStorage t'' locs s"
    using 14(4) unfolding copy_def using copySingleChange_typedPref[of p l t "(Storage st (Address
env))" x s] by simp
  then have a35:"∀ locs t''. locs ≠ l ∧ ¬ TypedStoSubpref locs l (STArray x t) ⟶
     accessStorage t'' locs (Storage st (Address env)) = accessStorage t'' locs (Storage st' (Address
env))"
    using 14 by auto
  have mInStd:"s = Storage st' (Address env)" using 14 by simp

  obtain stloc tp'' where MConsrc:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st⦇Gas
:= state.Gas st - costs
      (ASSIGN lv ex) env cd st⦈) (Address env)) ∧ (∃xx. KStoptr p = KStoptr xx) ∧

          (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue env) ∧
          (tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc ∨ extractValueType (KStoptr p) ≠ stloc ∧
CompStoType tp'' (STArray x t) stloc (extractValueType (KStoptr p)))"
    using 2(1) 14(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex)
env cd st⦈)"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KStoptr p" "type.Storage (STArray x t)" g]
    using 2(3)
    by (auto split:type.splits if_splits )

  note limitSt1 = TypeSafe_memory_bounds(1)[OF 2(1)]
  note limitSt = TypeSafe_memory_bounds(2)[OF 2(1)]
  moreover have allocateSame:"∀ loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))" using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
```

```
  have stackDenvalLimits:"∀struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                          ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
    using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto

  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof intros
    show "AddressTypes (Accounts st')" using 2(1) 14 unfolding TypeSafe_def by simp
  next
    show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
    show "compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
     (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
     (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧ (l2
= Stackloc l2'
         ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
      then show " if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "TypedStoSubpref stl2 stl1 tp1")
        case True
        then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref stl1 stl2 tp2")
          case True
          then show ?thesis using a1 sameStack a0 unfolding compPointers_def by simp
        next
          case False
          then show ?thesis using f1 by simp
        qed
      qed
    qed
  next
    have scOld:"safeContract (Accounts st) (Storage st)" using 2(1) unfolding TypeSafe_def by simp
    show "safeContract (Accounts st') (Storage st')" unfolding safeContract_def
    proof intros
      fix e ct dud i tp
      assume *:"Type (Accounts st' (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
               ep $$ Contract (e::environment) = Some (ct, dud) ∧
               ct $$ i = Some (Var tp)"
      show "SCon tp i (Storage st' (Address e))"
      proof (cases "Address e = Address env")
        case False
        then have sameAddr:"Storage st' (Address e) = Storage st (Address e)" using 14 by simp
        moreover have "SCon tp i (Storage st (Address e))"
          using * scOld 14 unfolding safeContract_def by fastforce
        ultimately show ?thesis by simp
      next
        case addrEq:True
        have typedOld:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
          using * 14 by simp
        have epOld:"ep $$ Contract e = Some (ct, dud)"
          using * by simp
        have ctOld:"ct $$ i = Some (Var tp)"
          using * by simp
        have denI:"Denvalue env $$ i = Some (type.Storage tp, Storeloc i)"
          using fi_contract_var_to_denvalue_storeloc[OF 2(3) addrEq typedOld epOld ctOld] .
        have inDenI:"(type.Storage tp, Storeloc i) |∈| fmran (Denvalue env)"
```

```
          using denI by (simp add: fmranI)
        have oldSConE:"SCon tp i (Storage st (Address e))"
          using scOld typedOld epOld ctOld unfolding safeContract_def by blast
        have oldSCon:"SCon tp i (Storage st (Address env))"
          using oldSConE addrEq by simp
        have newSCon:"SCon tp i (Storage st' (Address env))"
        proof(cases rule:lexpStorageG[OF 2(1) 14(2) 2(3)])
          case lInDen:1
          have cmpStoPtr:"(
            (type.Storage tp, l') |∈| fmran (Denvalue env) ∧
            (type.Storage (STArray x t), Storeloc l) |∈| fmran (Denvalue env) ∧
             l' = Storeloc i)
             ⟶
            (if TypedStoSubpref l i tp then CompStoType tp (STArray x t) i l
             else if TypedStoSubpref i l (STArray x t) then CompStoType (STArray x t) tp l i else True)"
            using 2(1) 14(3) lInDen inDenI unfolding TypeSafe_def compPointers_def by blast
          then have cmpStoPtr2:"(if TypedStoSubpref l i tp then CompStoType tp (STArray x t) i l
             else if TypedStoSubpref i l (STArray x t) then CompStoType (STArray x t) tp l i else True)"
            using 14(3) lInDen inDenI sameStack
            using 2(1) compPointers_def typeSafeCompPointers by blast

          have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14 lInDen
              sameStoLocTSafe by blast
          have p:"((if TypedStoSubpref stloc l t' then CompStoType t' tp'' l stloc
             else if TypedStoSubpref l stloc tp'' then CompStoType tp'' t' stloc l else True))"
            using 2(1) unfolding TypeSafe_def compPointers_def using lInDen MConsrc by blast
          have denSub:"(if TypedStoSubpref stloc l (STArray x t) then CompStoType (STArray x t) tp'' l
stloc
             else if TypedStoSubpref l stloc tp'' then CompStoType tp'' (STArray x t) stloc l else
True)"
            using p 14(3) by blast
          have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
                ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
            using MConsrc by simp
          have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
          have rel:"CompStoType t' (STArray x t) l l"
            using 14(3) CompStoType_sameLocNdTyp by simp
          have SCondest:"SCon (STArray x t) l s"
            using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf ] by auto
          then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
            by (simp add: mInStd)
          then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
            by (simp add: mInStd )
          show ?thesis
          proof(cases "TypedStoSubpref l i tp")
            case True
            then have "CompStoType tp (STArray x t) i l"
              by (simp add: cmpStoPtr2)
            then show ?thesis using SCondest2 oldSCon a35 TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref i l (STArray x t)")
              case True
              then have "CompStoType t' tp l i " using cmpStoPtr2 f1 14(3) by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs by blast
            next
              case False
              have notSame:"i ≠ l" using False by auto
              have k7:"∀locs. TypedStoSubpref locs i tp ⟶ locs ≠ l " using f1 by blast
              have k8:"∀locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ i" using False by auto
```

```
            show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 oldSCon False] by blast
          qed
        qed
      next
        case sub2:(2 l''' t''')
        have cmpStoPtr:"(
            (type.Storage tp, Storeloc i) |∈| fmran (Denvalue env) ∧
            (type.Storage t''', Storeloc l''') |∈| fmran (Denvalue env) )
             ⟶
            (if TypedStoSubpref l''' i tp then CompStoType tp t''' i l'''
             else if TypedStoSubpref i l''' t''' then CompStoType t''' tp l''' i else True)"
          using 2(1) 14(3) inDenI unfolding TypeSafe_def compPointers_def by blast
        then have cmpStoPtr2:"(if TypedStoSubpref l''' i tp then CompStoType tp t''' i l'''
             else if TypedStoSubpref i l''' t''' then CompStoType t''' tp l''' i else True)"
          using 14(3) sub2 inDenI sameStack by auto

        have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce
        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14 sameStoLocTSafe
sub2
          using SCon_imps_sublocs by blast
        have p:"((if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using sub2 MConsrc by blast
        have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
              ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
        have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
        have rel:"CompStoType t''' (STArray x t) l''' l" using sub2 14(3) by auto
        have SCondest:"SCon (STArray x t) l s"
          using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
        then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
          by (simp add: mInStd)
        then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
          by (simp add: mInStd )
        show ?thesis
        proof(cases "TypedStoSubpref l''' i tp")
          case True
          then have "CompStoType tp t''' i l'''" using cmpStoPtr2 by simp
          then have "CompStoType tp (STArray x t) i l" using sub2 using CompStoType_trns 14(3) by
blast
          then show ?thesis using SCondest2 oldSCon a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref i l''' t'''")
            case True
            then have "l''' ≠ i" using f1 using TypedStoSubpref_sameLoc by auto
            then show ?thesis
            proof(cases "l = i")
              case t5:True
              then have "CompStoType t''' tp l''' i" using cmpStoPtr2 f1 14(3) True by simp
              moreover have "CompStoType t''' t' l''' i" using t5 sub2 by auto
              ultimately have "t' = tp" using CompStoType_sameLocs_sameType by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
            next
              case False
```

```
              then have "CompStoType t''' tp l''' i" using cmpStoPtr2 f1 14(3) True by simp
              then show ?thesis
              proof(cases "CompStoType t' tp l i")
                case True
                then show ?thesis using SCon_imps_sublocs SCondest3 by blast
              next
                case False
                then show ?thesis
                  using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t''' tp l''' i>
a35 scl''' sub2(2) 14(3) by blast
              qed
            qed
          next
            case False
            then have asm10:"¬ TypedStoSubpref l i tp" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1] sub2 by blast
            then have asm20:"¬ TypedStoSubpref i l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
            have notSame:"i ≠ l" using False sub2 by blast
            have k7:"∀locs. TypedStoSubpref locs i tp ⟶ locs ≠ l " using f1 sub2 asm10 asm20 by
blast
            have k8:"∀locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ i" using False sub2
14(3) asm10 asm20 by blast
            show ?thesis using sublocs_nonchanged_SCon[OF _ a35 oldSCon] asm10 asm20 14(3) by blast
          qed
        qed
      next
        case sub3:(3 l''' t''' l'''')
        have cmpStoPtr:"(
            (type.Storage tp, Storeloc i) |∈| fmran (Denvalue env) ∧
            (type.Storage t''', Stackloc l'''') |∈| fmran (Denvalue env) ∧
             accessStore l'''' (Stack st) = Some (KStoptr l'''))
             ⟶
            (if TypedStoSubpref l''' i tp then CompStoType tp t''' i l'''
             else if TypedStoSubpref i l''' t''' then CompStoType t''' tp l''' i else True)"
          using 2(1) 14(3) inDenI unfolding TypeSafe_def compPointers_def by blast
        then have cmpStoPtr2:"(if TypedStoSubpref l''' i tp then CompStoType tp t''' i l'''
             else if TypedStoSubpref i l''' t''' then CompStoType t''' tp l''' i else True)"
          using 14(3) sub3 inDenI sameStack by auto

        have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce
        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14 sameStoLocTSafe
sub3
          using SCon_imps_sublocs by blast
        have p:"(
            (if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using sub3 MConsrc by blast
        have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
              ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
        have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
        have rel:"CompStoType t''' (STArray x t) l''' l" using sub3 14(3) by auto
        have SCondest:"SCon (STArray x t) l s"
          using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
        then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
          by (simp add: mInStd)
        then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
```

```
          by (simp add: mInStd )
        show ?thesis
        proof(cases "TypedStoSubpref l''' i tp")
          case True
          then have "CompStoType tp t''' i l''''" using cmpStoPtr2 by simp
          then have "CompStoType tp (STArray x t) i l" using sub3 using CompStoType_trns 14(3) by
blast
          then show ?thesis using SCondest2 oldSCon a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref i l''' t''''")
            case True
            then have "l''' ≠ i" using f1 using TypedStoSubpref_sameLoc by auto
            then show ?thesis
            proof(cases "l = i") case t5:True
              then have "CompStoType t''' tp l''' i" using cmpStoPtr2 f1 14(3) True by simp
              moreover have "CompStoType t''' t' l''' i" using t5 sub3 by auto
              ultimately have "t' = tp" using CompStoType_sameLocs_sameType by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
            next
              case False
              then have "CompStoType t''' tp l''' i" using cmpStoPtr2 f1 14(3) True by simp
              then show ?thesis
              proof(cases "CompStoType t' tp l i")
                case True
                then show ?thesis using SCon_imps_sublocs SCondest3 by blast
              next
                case False
                then show ?thesis
                  using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t''' tp l''' i›
a35 scl''' sub3(3) 14(3) by blast
              qed
            qed
          next
            case False
            then have asm10:"¬ TypedStoSubpref l i tp" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1] sub3 by blast
            then have asm20:"¬ TypedStoSubpref i l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
            have notSame:"i ≠ l" using False sub3 by blast
            have k7:"∀ locs. TypedStoSubpref locs i tp ⟶ locs ≠ l " using f1 sub3 asm10 asm20 by
blast
            have k8:"∀ locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ i" using False sub3
14(3) asm10 asm20 by blast
            show ?thesis using sublocs_nonchanged_SCon[OF _ a35 oldSCon] asm10 asm20 14(3) by blast
          qed
        qed
      qed
      then show ?thesis using newSCon addrEq by simp

    qed
  qed
  next
    show "balanceTypes (Accounts st')" using 14 using 2(1) unfolding TypeSafe_def by simp
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
  next
    have *:"((∀ tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore loc
```

```
(Stack st) = None) ∧
           (∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc)))) "
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    have **:"Toploc (Stack st) = Toploc (Stack st')" using 14 unfolding updateStore_def by auto
    show "lessThanTopLocs (Stack st')" using sameStack * ** unfolding lessThanTopLocs_def by auto
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
  next
    show "lessThanTopLocs (Memory st')" using sameMemory unfolding lessThanTopLocs_def
      by (simp add: limitSt limitSt1)
  next
    show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t'' l'
      assume inDen:" (t'', l') |∈| fmran (Denvalue env)"

      show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t'' l'"
      proof(cases l')
        case (Stackloc x1)
        then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1) un-
folding TypeSafe_def typeCompat_def using sameStack by fastforce
        then show ?thesis
        proof(cases "a")
          case (KValue x1)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5) stackvalue.simps(17)
)
        next
          case (KCDptr x2)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack unfolding TypeSafe_def
typeCompat_def
            by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(18))
        next
          case (KMemptr x3)
          then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack sameMemory unfolding
TypeSafe_def typeCompat_def by (cases t''; fastforce)
        next
          case (KStoptr x4)
          then obtain struct where structDef: "t'' = type.Storage struct" using Stackloc adef inDen
Stackloc 2(1) sameStack unfolding TypeSafe_def typeCompat_def
            by (cases t''; fastforce)

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases rule:lexpStorageG[OF 2(1) 14(2) 2(3)])
            case lInDen:1
            then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14
                sameStoLocTSafe by blast

            have p:"((if TypedStoSubpref stloc l t' then CompStoType t' tp'' l stloc
               else if TypedStoSubpref l stloc tp'' then CompStoType tp'' t' stloc l else True))"
              using 2(1) unfolding TypeSafe_def compPointers_def using lInDen MConsrc by blast

            have denSub:"(if TypedStoSubpref stloc l (STArray x t) then CompStoType (STArray x t) tp''
l stloc
               else if TypedStoSubpref l stloc tp'' then CompStoType tp'' (STArray x t) stloc l else
True)"
              using p 14(3) by blast
            have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
                 ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
```

```
        using MConsrc by simp
      have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
      have rel:"CompStoType t' (STArray x t) l l"
        using 14(3) CompStoType_sameLocNdTyp by simp
      have SCondest:"SCon (STArray x t) l s"
        using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf] by simp

      then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
        by (simp add: mInStd)
      then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
        by (simp add: mInStd )

      have cmpStoPtr:"(
        (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
        (type.Storage (STArray x t), Storeloc l) |∈| fmran (Denvalue env) ∧
         l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
         ⟶
        (if TypedStoSubpref l x4 struct then CompStoType struct (STArray x t) x4 l
         else if TypedStoSubpref x4 l (STArray x t) then CompStoType (STArray x t) struct l x4
else True)"
        using 2(1) 14(3) lInDen inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

      then have cmpStoPtr2:"(if TypedStoSubpref l x4 struct then CompStoType struct (STArray x t)
x4 l
         else if TypedStoSubpref x4 l (STArray x t) then CompStoType (STArray x t) struct l x4
else True)"
        using 14(3) lInDen inDen adef Stackloc KStoptr structDef sameStack by auto

      have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
        by fastforce

      have "SCon struct x4 (Storage st' (Address env))"
      proof(cases "TypedStoSubpref l x4 struct")
        case True
        then have "CompStoType struct (STArray x t) x4 l" using cmpStoPtr2 by simp
        then show ?thesis using SCondest2 SConx4Old a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
          by blast
      next
        case f1:False
        then show ?thesis
        proof(cases "TypedStoSubpref x4 l (STArray x t)")
          case True
          then have "CompStoType t' struct l x4 " using cmpStoPtr2 f1 14(3) by simp
          then show ?thesis using SCondest3 SCon_imps_sublocs by blast
        next
          case False
          have notSame:"x4 ≠ l" using False by auto
          have k7:"∀locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 by blast
          have k8:"∀locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ x4" using False by
auto
          show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
        qed
      qed
      then show ?thesis by simp
    next
      case sub2:(2 l''' t''')
      have cmpStoPtr:"(
        (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
        (type.Storage t''', Storeloc l''') |∈| fmran (Denvalue env) ∧
         l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4))
         ⟶
```

```
            (if TypedStoSubpref l''' x4 struct then CompStoType struct t''' x4 l'''
             else if TypedStoSubpref x4 l''' t''' then CompStoType t''' struct l''' x4 else True)"
            using 2(1) 14(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

          then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t''' x4
l'''
             else if TypedStoSubpref x4 l''' t''' then CompStoType t''' struct l''' x4 else True)"
            using 14(3) sub2 inDen adef Stackloc KStoptr structDef sameStack by auto

          have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
            by fastforce
          have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

          then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14
              sameStoLocTSafe sub2
            using SCon_imps_sublocs by blast

          have p:"(
            (if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
            using 2(1) unfolding TypeSafe_def compPointers_def using sub2 MConsrc by blast

          have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
            using p 14(3) by blast
          have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
               ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
            using MConsrc by simp
          have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
          have rel:"CompStoType t''' (STArray x t) l''' l" using sub2 14(3) by auto
          have SCondest:"SCon (STArray x t) l s"
            using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
          then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
            by (simp add: mInStd)
          then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
            by (simp add: mInStd )

          have "SCon struct x4 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x4 struct")
            case True
            then have "CompStoType struct t''' x4 l'''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t) x4 l" using sub2
              using CompStoType_trns 14(3) by blast
            then show ?thesis using SConx4Old a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent SCondest2 by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x4 l''' t")
              case True
              then have "l''' ≠ x4" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x4")
                case t5:True
                then have " CompStoType t''' struct l''' x4" using cmpStoPtr2 f1 14(3) True
                  using sub2(2) by auto
                moreover have "CompStoType t''' t' l''' x4" using t5 sub2 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
```

```
                by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
            next
              case False
              then show ?thesis
              proof(cases "CompStoType t' struct l x4 ")
                case True
                then show ?thesis
                  using SCon_imps_sublocs SCondest3 by blast
              next
                case False
                then show ?thesis
                  using SCon_preserved_array_update[OF rel cmpStoPtr2 f1 SCondest2 SConx4Old a35]
by blast
              qed
            qed
          next
            case False
            show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] 14(3)
              using SCon_preserved_array_update[OF rel cmpStoPtr2 f1 SCondest2 SConx4Old a35] by
blast
          qed
        qed
        then show ?thesis by simp
      next
        case sub3:(3 l''' t''' l'''')
        have cmpStoPtr:"(
          (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
          (type.Storage t''', Stackloc l'''') |∈| fmran (Denvalue env) ∧
           l' = Stackloc x1 ∧ accessStore x1 (Stack st) = Some (KStoptr x4)) ∧ accessStore l''''
(Stack st) = Some (KStoptr l''')
            ⟶
          (if TypedStoSubpref l''' x4 struct then CompStoType struct t''' x4 l'''
           else if TypedStoSubpref x4 l''' t''' then CompStoType t''' struct l''' x4 else True)"
          using 2(1) 14(3) inDen adef Stackloc KStoptr structDef unfolding TypeSafe_def
compPointers_def by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l''' x4 struct then CompStoType struct t''' x4
l'''
           else if TypedStoSubpref x4 l''' t''' then CompStoType t''' struct l''' x4 else True)"
          using 14(3) sub3 inDen adef Stackloc KStoptr structDef sameStack by auto

        have SConx4Old:"SCon struct x4 (Storage st (Address env))" using 2(1) unfolding
TypeSafe_def typeCompat_def using adef Stackloc KStoptr inDen 2(1) sameStack structDef
          by fastforce
        have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14
            sameStoLocTSafe sub3
          using SCon_imps_sublocs by blast

        have p:"((if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using sub3 MConsrc by blast

        have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
            ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
```

```
            have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
            have rel:"CompStoType t''' (STArray x t) l''' l" using sub3 14(3) by auto
            have SCondest:"SCon (STArray x t) l s"
              using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
            then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
              by (simp add: mInStd)
            then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
              by (simp add: mInStd )

            have "SCon struct x4 (Storage st' (Address env))"
            proof(cases "TypedStoSubpref l''' x4 struct")
              case True
              then have "CompStoType struct t''' x4 l''''" using cmpStoPtr2 by simp
              then have "CompStoType struct (STArray x t) x4 l" using sub3
                using CompStoType_trns 14(3) by blast
              then show ?thesis using SCondest2 SConx4Old a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
                by blast
            next
              case f1:False
              then show ?thesis
              proof(cases "TypedStoSubpref x4 l''' t''''")
                case True
                then have "l''' ≠ x4" using f1
                  using TypedStoSubpref_sameLoc by auto
                then show ?thesis
                proof(cases "l = x4")
                  case t5:True
                  then have " CompStoType t''' struct l''' x4" using cmpStoPtr2 f1 14(3) True by simp
                  moreover have "CompStoType t''' t' l''' x4" using t5 sub3 by auto
                  ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                    by simp
                  then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
                next
                  case False
                  then have " CompStoType t''' struct l''' x4" using cmpStoPtr2 f1 14(3) True by simp
                  then show ?thesis
                  proof(cases "CompStoType t' struct l x4 ")
                    case True
                    then show ?thesis
                      using SCon_imps_sublocs SCondest3 by blast
                  next
                    case False
                    then show ?thesis
                      using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t''' struct
l''' x4› a35 scl''' sub3(3) 14(3) by blast
                  qed
                qed
              next
                case False
                then have asm10:"¬ TypedStoSubpref l x4 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
                then have asm20:"¬ TypedStoSubpref x4 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
                have notSame:"x4 ≠ l" using False sub3 by blast
                have k7:"∀ locs. TypedStoSubpref locs x4 struct ⟶ locs ≠ l " using f1 sub3 asm10
asm20 by blast
                have k8:"∀ locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ x4" using False sub3
14(3) asm10 asm20 by blast
                show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 14(3) by
blast
              qed
            qed
            then show ?thesis by simp
```

```
        qed
        then show ?thesis using Stackloc inDen 2(1) sameStack KStoptr adef structDef by simp
      qed
    next
      case (Storeloc x2)
      then obtain struct where structDef: "t'' = type.Storage struct" using Storeloc inDen 2(1)
sameStack unfolding TypeSafe_def typeCompat_def
        by (cases t''; fastforce)

      have "SCon struct x2 (Storage st' (Address env))"
      proof(cases rule:lexpStorageG[OF 2(1) 14(2) 2(3)])
        case lInDen:1
        have cmpStoPtr:"(
          (type.Storage struct, l') |∈| fmran (Denvalue env) ∧
          (type.Storage (STArray x t), Storeloc l) |∈| fmran (Denvalue env) ∧
           l' = Storeloc x2)
           ⟶
          (if TypedStoSubpref l x2 struct then CompStoType struct (STArray x t) x2 l
           else if TypedStoSubpref x2 l (STArray x t) then CompStoType (STArray x t) struct l x2 else
True)"
          using 2(1) 14(3) lInDen inDen Storeloc structDef unfolding TypeSafe_def compPointers_def
by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l x2 struct then CompStoType struct (STArray x t)
x2 l
           else if TypedStoSubpref x2 l (STArray x t) then CompStoType (STArray x t) struct l x2 else
True)"
          using 14(3) lInDen inDen Storeloc structDef sameStack by simp

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce

        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14 lInDen
            sameStoLocTSafe by blast

        have p:"((if TypedStoSubpref stloc l t' then CompStoType t' tp'' l stloc
           else if TypedStoSubpref l stloc tp'' then CompStoType tp'' t' stloc l else True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using lInDen MConsrc by blast

        have denSub:"(if TypedStoSubpref stloc l (STArray x t) then CompStoType (STArray x t) tp'' l
stloc
           else if TypedStoSubpref l stloc tp'' then CompStoType tp'' (STArray x t) stloc l else
True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
              ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
        have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
        have rel:"CompStoType t' (STArray x t) l l"
          using 14(3) CompStoType_sameLocNdTyp by simp
        have SCondest:"SCon (STArray x t) l s"
          using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf ] by simp

        then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
          by (simp add: mInStd)
        then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
          by (simp add: mInStd )

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases "TypedStoSubpref l x2 struct")
          case True
          then have "CompStoType struct (STArray x t) x2 l"
```

```
            by (simp add: cmpStoPtr2)
          then show ?thesis using SCondest2 SConx4Old a35 TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref x2 l (STArray x t)")
            case True
            then have "CompStoType t' struct l x2 " using cmpStoPtr2 f1 14(3) by simp
            then show ?thesis using SCondest3 SCon_imps_sublocs by blast
          next
            case False
            have notSame:"x2 ≠ l" using False by auto
            have k7:"∀locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 by blast
            have k8:"∀locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ x2" using False by auto
            show ?thesis using sublocs_nonchanged_SCon[OF f1 a35 SConx4Old False] by blast
          qed
        qed
        then show ?thesis by simp
      next
        case sub2:(2 l''' t''')
        have cmpStoPtr:"(
            (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
            (type.Storage t''', Storeloc l''') |∈| fmran (Denvalue env) )
             ⟶
            (if TypedStoSubpref l''' x2 struct then CompStoType struct t''' x2 l'''
             else if TypedStoSubpref x2 l''' t''' then CompStoType t''' struct l''' x2 else True)"
          using 2(1) 14(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t''' x2 l'''
             else if TypedStoSubpref x2 l''' t''' then CompStoType t''' struct l''' x2 else True)"
          using 14(3) sub2 inDen Storeloc structDef sameStack by auto

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce
        have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub2 by fastforce

        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14
            sameStoLocTSafe sub2
          using SCon_imps_sublocs by blast

        have p:"((if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using sub2 MConsrc by blast

        have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
              ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
        have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
        have rel:"CompStoType t''' (STArray x t) l''' l" using sub2 14(3) by auto
        have SCondest:"SCon (STArray x t) l s"
          using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
        then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
          by (simp add: mInStd)
        then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
          by (simp add: mInStd )
```

```
          have "SCon struct x2 (Storage st' (Address env))"
          proof(cases "TypedStoSubpref l''' x2 struct")
            case True
            then have "CompStoType struct t''' x2 l''''" using cmpStoPtr2 by simp
            then have "CompStoType struct (STArray x t) x2 l" using sub2
              using CompStoType_trns 14(3) by blast
            then show ?thesis using SCondest2 SConx4Old a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
              by blast
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref x2 l''' t''''")
              case True
              then have "l''' ≠ x2" using f1
                using TypedStoSubpref_sameLoc by auto
              then show ?thesis
              proof(cases "l = x2")
                case t5:True
                then have " CompStoType t''' struct l''' x2" using cmpStoPtr2 f1 14(3) True by simp
                moreover have "CompStoType t''' t' l''' x2" using t5 sub2 by auto
                ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                  by simp
                then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
              next
                case False
                then have " CompStoType t''' struct l''' x2" using cmpStoPtr2 f1 14(3) True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x2 ")
                  case True
                  then show ?thesis using SCon_imps_sublocs SCondest3 by blast
                next
                  case False
                  then show ?thesis
                    using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 ‹CompStoType t''' struct
l''' x2› a35 scl''' sub2(2) 14(3) by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub2 by blast
              then have asm20:"¬ TypedStoSubpref x2 l t'" using sub2 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x2 ≠ l" using False sub2 by blast
              have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub2 asm10 asm20
by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ x2" using False sub2
14(3) asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 14(3) by
blast
            qed
          qed

          then show ?thesis by simp
        next
          case sub3:(3 l''' t''' l'''')
          have cmpStoPtr:"(
              (type.Storage struct, Storeloc x2) |∈| fmran (Denvalue env) ∧
              (type.Storage t''', Stackloc l'''') |∈| fmran (Denvalue env) ∧
               accessStore l'''' (Stack st) = Some (KStoptr l'''))
               ⟶
```

```
            (if TypedStoSubpref l''' x2 struct then CompStoType struct t''' x2 l'''
             else if TypedStoSubpref x2 l''' t''' then CompStoType t''' struct l''' x2 else True)"
          using 2(1) 14(3) inDen Storeloc structDef unfolding TypeSafe_def compPointers_def by blast

        then have cmpStoPtr2:"(if TypedStoSubpref l''' x2 struct then CompStoType struct t''' x2 l'''
             else if TypedStoSubpref x2 l''' t''' then CompStoType t''' struct l''' x2 else True)"
          using 14(3) sub3 inDen Storeloc structDef 14(3) sameStack by auto

        have SConx4Old:"SCon struct x2 (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using Storeloc inDen 2(1) sameStack structDef
          by fastforce
        have scl''':"SCon t''' l''' (Storage st (Address env))" using 2(1) unfolding TypeSafe_def
typeCompat_def using sub3 by fastforce

        then have h7:"SCon (STArray x t) l (Storage st (Address env))" using 2(1) 14
            sameStoLocTSafe sub3
          using SCon_imps_sublocs by blast

        have p:"(
            (if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
             else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else
True))"
          using 2(1) unfolding TypeSafe_def compPointers_def using sub3 MConsrc by blast

        have denSub:"(if TypedStoSubpref stloc l''' t''' then CompStoType t''' tp'' l''' stloc
           else if TypedStoSubpref l''' stloc tp'' then CompStoType tp'' t''' stloc l''' else True)"
          using p 14(3) by blast
        have paInf:"(tp'' = (STArray x t) ∧ KStoptr p = KStoptr stloc
              ∨ extractValueType (KStoptr p) ≠ stloc ∧ CompStoType tp'' (STArray x t) stloc
(extractValueType (KStoptr p)))"
          using MConsrc by simp
        have h8:"SCon (STArray x t) p (Storage st (Address env))" using MConsrc by simp
        have rel:"CompStoType t''' (STArray x t) l''' l" using sub3 14(3) by auto
        have SCondest:"SCon (STArray x t) l s"
          using copy_dest_scon_from_rel[OF 14(4) 14(3) h7 h8 denSub paInf rel] .
        then have SCondest2:"SCon (STArray x t) l (Storage st' (Address env))"
          by (simp add: mInStd)
        then have SCondest3:"SCon t' l (Storage st' (Address env))" using 14(3)
          by (simp add: mInStd )

        have "SCon struct x2 (Storage st' (Address env))"
        proof(cases "TypedStoSubpref l''' x2 struct")
          case True
          then have "CompStoType struct t''' x2 l'''" using cmpStoPtr2 by simp
          then have "CompStoType struct (STArray x t) x2 l" using sub3
            using CompStoType_trns 14(3) by blast
          then show ?thesis using SCondest2 SConx4Old a35 14(3) TypedStoSubpref_shared_parent_related
SCon_sub_imps_Parent
            by blast
        next
          case f1:False
          then show ?thesis
          proof(cases "TypedStoSubpref x2 l''' t'''")
            case True
            then have "l''' ≠ x2" using f1
              using TypedStoSubpref_sameLoc by auto
            then show ?thesis
            proof(cases "l = x2")
              case t5:True
              then have " CompStoType t''' struct l''' x2" using cmpStoPtr2 f1 14(3) True by simp
              moreover have "CompStoType t''' t' l''' x2" using t5 sub3 by auto
              ultimately have "t' = struct" using CompStoType_sameLocs_sameType
                by simp
              then show ?thesis using SCondest3 SCon_imps_sublocs t5 by simp
```

```
              next
                case False
                then have " CompStoType t''' struct l''' x2" using cmpStoPtr2 f1 14(3) True by simp
                then show ?thesis
                proof(cases "CompStoType t' struct l x2 ")
                  case True
                  then show ?thesis
                    using SCon_imps_sublocs SCondest3 by blast
                next
                  case False
                  then show ?thesis
                    using SCon_imps_sublocs SCon_sub_imps_Parent SCondest2 <CompStoType t''' struct
l''' x2> a35 scl''' sub3(3) 14(3) by blast
                qed
              qed
            next
              case False
              then have asm10:"¬ TypedStoSubpref l x2 struct" using NotRelatedPrnt_imps_notRelatedChild[OF
False f1 ] sub3 by blast
              then have asm20:"¬ TypedStoSubpref x2 l t'" using sub3 NotReachablePrnt_imps_notReachableChild
False f1 by blast
              have notSame:"x2 ≠ l" using False sub3 by blast
              have k7:"∀ locs. TypedStoSubpref locs x2 struct ⟶ locs ≠ l " using f1 sub3 asm10 asm20
by blast
              have k8:"∀ locs. TypedStoSubpref locs l (STArray x t) ⟶ locs ≠ x2" using False sub3
14(3) asm10 asm20 by blast
              show ?thesis using sublocs_nonchanged_SCon[OF _ a35 SConx4Old] asm10 asm20 14(3) by
blast
            qed
          qed
          then show ?thesis by simp
        qed


        then show ?thesis using Storeloc inDen 2(1) sameStack Storeloc structDef by simp
      qed
    qed
    then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd " us-
ing sameStack by simp
  next
    have "Accounts st'= Accounts st" using 14 by auto
    then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) unfolding
fullyInitialised_def
      using sameStack by simp
  next
    have "Stack st = Stack st'" using 14 by simp
    moreover have "Memory st = Memory st'" using 14 by auto
    ultimately show "denvalueTypeCorrectness env (Stack st') (Memory st')"
      using 2(1) unfolding TypeSafe_def by simp
  next
    show "subPrefixStructuralConsistency (Memory st')"
      using 2(1) 14 unfolding TypeSafe_def by simp
  next
    show " SomeValSomeTyp (Memory st') " using 2(1) unfolding TypeSafe_def using 14 by simp
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
      using 2(1) 14 unfolding TypeSafe_def by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 14 by auto
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
```

```
st') = Some (MValue v')"
      using 14 by auto
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 14 by auto
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 14 by auto
  qed
qed
```

Assignment case 15: a storage array assigned to a memory location: the array is copied into fresh memory and the location is updated to point to it.

```
lemma TypeSafe_Assign_15:
  assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
    and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
    and "fullyInitialised env (Accounts st) (Stack st)"
    and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
        ex) env cd st) = Normal ((KStoptr p, type.Storage (STArray x t)), g)"
    and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LMemloc l, type.Memory t'),g')"
    and "cps2mTypeCompatible (STArray x t) t'"
    and "cps2m p (ShowL_nat (Toploc (Memory st))) x t (Storage st (Address env)) (snd (allocate(Memory
st))) = Some m"
    and "updateStore l (MPointer ((ShowL_nat (Toploc (Memory st))))) m = m'"
    and "st' = st(|state.Gas := g', Memory := m'|)"
  shows "StateInvariant env st st' cd"
proof -
  let ?case = "StateInvariant env st st' cd"
  note 2 = assms(1-3)
  note 15 = assms(4-9)
  have temp:"TypeSafe env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|))) (Memory (st(|Gas := g|)))
(Storage (st(|Gas := g|))) cd"
    using 2(1) by simp
  have ttt:"fullyInitialised env (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|)))" using 2(3) unfold-
ing fullyInitialised_def by simp

  then show ?thesis
  proof(cases rule:lexpIndexMem[OF 15(2) temp ttt])
    case lInfo:(1 x21 x22 tp tParent l' l'' prnt len' arr' i)
    have nonChangedStack:"∀ loc. loc ≠ l ⟶ accessStore loc (Stack st) = accessStore loc (Stack st')"
using 15 unfolding accessStore_def updateStore_def by auto
    have sameStack:"(Stack st') = Stack st" using 15 unfolding accessStore_def updateStore_def by auto
    have sameStorage:"Storage st' = Storage st " using 15 by auto
    have nonLocChanged:"∀ locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
        (ShowL_nat (Toploc (Memory st)))⟶ accessStore locs (snd (allocate (Memory st))) = accessStore
        locs m"
      using 15(4) unfolding cps2m_def using cps2mSingleChange[of p "(ShowL_nat (Toploc (Memory st)))" t
"(Storage st
        (Address env))" " (snd (allocate (Memory st)))" x m ] cps2m_def[of p "(ShowL_nat (Toploc
(Memory st)))" x
        t "(Storage st (Address env))" " (snd (allocate (Memory st)))" ]
      by fastforce
    have a30:"∀ locs. locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ TypedStoSubpref locs
        (ShowL_nat (Toploc (Memory st))) (STArray x t) ⟶ accessStore locs (snd (allocate (Memory st)))
        = accessStore locs m"
      using 15(4) unfolding cps2m_def using cps2mSingleChange_typedPref[of p "(ShowL_nat (Toploc (Memory
st)))" t "(Storage
        st (Address env))" " (snd (allocate (Memory st)))" x m ] by fastforce
    then have a32:"∀ locs. ¬ TypedStoSubpref locs (ShowL_nat (Toploc (Memory st))) (STArray x t) ⟶
accessStore locs (Memory st) = accessStore locs m"
      by (metis allocateSameAccess nonLocChanged)
    have selfPoint:"∀ l l'. l ≠ (ShowL_nat (Toploc (Memory st))) ∧ TypedStoSubpref l
        (ShowL_nat (Toploc (Memory st))) (STArray x t) ∧ accessStore l m = Some (MPointer l') ⟶
```

```
      l' = l"
    using 15(4) unfolding cps2m_def
    using cps2mSelfPointers[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))" "
(snd (allocate (Memory st)))" x m ]
    by blast
  have mInStd:"m' = Memory st'" using 15 by simp
  have NonChangeM'm:"∀ locs. locs ≠ l ⟶ accessStore locs m = accessStore locs m'" using 15(5)
unfolding accessStore_def updateStore_def by auto
  have accessL:"accessStore l m' = Some (MPointer (ShowL_nat (Toploc (Memory st))))" using 15(5) un-
folding accessStore_def updateStore_def by auto

  have MConsrc:"SCon (STArray x t) (extractValueType (KStoptr p)) (Storage (st⦇Gas := state.Gas st -
costs (ASSIGN lv ex) env cd
      st⦈)) (Address env)) ∧ (∃ xx. KStoptr p = KStoptr xx)"
    using 2(1) 15(1) exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv
ex) env cd st⦈)"
        "(state.Gas st - costs (ASSIGN lv ex) env cd st)" ] 2(3)unfolding fullyInitialised_def
    by (auto split:type.splits if_splits )

  obtain memArr where memArrDef: "t' = MTArray x memArr" using 15(3) cps2mTypeCompatible.simps
    by (metis mtypes.exhaust)

  have limitSt1:"(∀ loc y. accessStore loc (Memory st) = Some y ⟶ (∃ tloc<Toploc (Memory st).
LSubPrefL2 loc (ShowL_nat tloc)))"
    using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  have limitSt:"(∀ tloc loc. Toploc (Memory st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Memory st) = None)"
    using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
  moreover have allocateSame:"∀ loc. accessStore loc (Memory st) = accessStore loc (snd (allocate
(Memory st)))"
    using allocateSameAccess by blast
  ultimately have "accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None"
using LSubPrefL2_def by auto
  then have MCondest:" MCon (MTArray x memArr) m (ShowL_nat (Toploc (Memory st)))" using 15(4,3) un-
folding cps2m_def
    using cps2m[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))" "(snd (allocate
(Memory st)))" x m memArr]
      MConsrc extractValueType.simps(2) memArrDef by auto
  have "∃ x. accessStore l (Memory st) = Some x" using lInfo by (auto split:option.splits)
  then have l_not_toploc_orSub:"¬LSubPrefL2 l (ShowL_nat (Toploc (Memory st)))"
    using limitSt by fastforce

  have selfPoint2:"∀ l1 l2. LSubPrefL2 l1 (ShowL_nat (Toploc (Memory st)))
                  ∧ accessStore l1 (Memory st') = Some (MPointer l2) ⟶ l2 = l1 ∧ l1 ≠ (ShowL_nat
(Toploc (Memory st)))"
    by (metis (no_types, lifting) "15"(4) NonChangeM'm
        ‹accessStore (ShowL_nat (Toploc (state.Memory st))) (snd (allocate (state.Memory st))) = None›
a32 cps2mSelfPointers cps2m_def
        dual_order.refl l_not_toploc_orSub limitSt mInStd nonLocChanged option.discI)


  obtain len subT where tParentType:"tParent = MTArray len subT" using lInfo by blast
  then obtain p'' where lOrigin:"accessStore l (Memory (st)) = Some (MPointer p'')"
      and compType:"CompMemType (Memory (st⦇Gas := g⦈)) len subT (MTArray x memArr) l'' p''"
      and lsublocs:"l = hash prnt (ShowL_nat i) ∧ i < len' ∧ arr' = (MTArray x memArr) ∧ MCon
(MTArray len' arr') (Memory (st)) prnt"
      and lsublocs3:"(prnt = l'' ∧ len = len' ∧ arr' = subT ∨ CompMemType (Memory (st)) len subT
(MTArray len' arr') l'' prnt)"
    using lInfo 15(3) memArrDef by force
  then have lsublocs2:" CompMemType (Memory (st)) len' arr' (MTArray x memArr) prnt p''"
    using "15"(3) CompMemType.simps(2) memArrDef by blast

  then have bb9:"∀ subT subloc. CompMemType (Memory (st)) len' arr' subT prnt subloc ∧ subloc = p''
                          ⟶ subT = (MTArray x memArr)"
```

```
      using CompMemTypeSameLocsSameType lsublocs by blast

    have mconPrnt:"MCon (MTArray len' (MTArray x memArr)) (Memory st) prnt" using lsublocs by auto
    have ldef:"l = hash prnt (ShowL_nat i) ∧ i < len'" using lsublocs by auto

    have t6:"(type.Memory tParent, Stackloc l') |∈| fmran (Denvalue env)" using lInfo by blast
    have t7:" MCon (MTArray len subT) (Memory (st)) l''" using lInfo
      using tParentType by auto
    then have t8:"¬ LSubPrefL2 l'' (ShowL_nat (Toploc (Memory st)))" using limitSt limitSt1
typeSafe_noDenElementOverToploc_mem[OF 2(1) t6] lInfo(3) by simp
    have comptype2:"CompMemType m len subT (MTArray x memArr) l'' p''" using
        cpm2mCompMemTypeOld_imps_CompMemType[of "(Memory (st(|Gas := g|)))" len subT "(MTArray x memArr)"
l'' p'' "(ShowL_nat (Toploc (Memory st)))" m ]
        compType nonLocChanged limitSt t7 t8 allocateSame by auto
    have t10:"CompMemType (Memory st) len subT (MTArray x memArr) l'' p''"
      using
        cpm2mCompMemTypeOld_imps_CompMemType[of "(Memory (st(|Gas := g|)))" len subT "(MTArray x memArr)"
l'' p'' "(ShowL_nat (Toploc (Memory st)))" m ]
        compType nonLocChanged limitSt t7 t8 allocateSame by auto
    then have mconlOld:"MCon (MTArray x memArr) (Memory st) p''" using memArrDef 15(3) lInfo lOrigin
by auto

    have nonLocChanged2:"∀ locs. locs ≠ l ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st)))
        ⟶ accessStore locs (Memory st) = accessStore locs (Memory st')"
      using 15 nonLocChanged LSubPrefL2_def NonChangeM'm mInStd allocateSame by metis
    have stackDenvalLimits:"∀ struct loc stloc. (type.Memory struct, Stackloc loc) |∈| fmran (Denvalue
env)
                        ∧ accessStore loc (Stack st) = Some (KMemptr stloc) ⟶ ¬ LSubPrefL2 stloc
(ShowL_nat (Toploc (Memory st)))"
      using typeSafe_noDenElementOverToploc_mem[OF 2(1)] by auto
    have selfPoint_root:"∀ l1 l2. TypedStoSubpref l1 (ShowL_nat (Toploc (Memory st))) (STArray x t) ∧
accessStore l1 m = Some (MPointer l2) ⟶ l2 = l1"
    proof (intro allI impI)
      fix l1 l2
      assume asm:"TypedStoSubpref l1 (ShowL_nat (Toploc (Memory st))) (STArray x t) ∧ accessStore l1 m
= Some (MPointer l2)"
      have l1_neq:"l1 ≠ (ShowL_nat (Toploc (Memory st)))"
        using asm TypedStoSubpref_imp_LSubPrefL2 hash_inequality
        using ‹accessStore (ShowL_nat (Toploc (state.Memory st))) (snd (allocate (state.Memory st))) =
None› nonLocChanged
        by fastforce
      then show "l2 = l1"
        using selfPoint asm by blast
    qed
    have MCondest2:" MCon (MTArray x memArr) (Memory st') (ShowL_nat (Toploc (Memory st))) " using 15
MCondest memArrDef
    proof -
      have "¬ LSubPrefL2 (ShowL_nat (Toploc (Memory st))) l"
        using l_not_toploc_orSub MemLSubPrefTransitive LSubPrefL2_refl by blast
      moreover have "∀ d. ¬ LSubPrefL2 d l ⟶ accessStore d m = accessStore d (Memory st')"
        using NonChangeM'm mInStd LSubPrefL2_refl by metis
      moreover have "cps2mTypeCompatible (STArray x t) (MTArray x memArr)"
        using 15(3) memArrDef by simp
      ultimately show ?thesis
        using MCon_preserved_under_disjoint_location[of "(ShowL_nat (Toploc (Memory st)))" l "STArray x
t" m
            "MTArray x memArr" "Memory st'"] selfPoint_root l_not_toploc_orSub MCondest by blast
    qed

    have bb:"∀ locs tp. CompMemType (Memory st) len subT tp l'' locs ⟶ locs ≠ (ShowL_nat
        (Toploc (Memory st))) ∧ ¬LSubPrefL2 locs (ShowL_nat (Toploc (Memory st)))"
    proof intros
      fix locs tp
      assume asm1:"CompMemType (Memory st) len subT tp l'' locs"
```

```
      then have a2:"locs ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc
(Memory st)))" using t7
      proof(induction subT arbitrary:len l'')
        case (MTArray x1 subT)
        then show ?case
          using CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def limitSt limitSt1 subPtrs_nonTop by
blast
      next
        case (MTValue x)
        then show ?case
          by (metis (no_types, lifting) CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def limitSt
limitSt1 subPtrs_nonTop)
      qed
      then show " locs ≠ (ShowL_nat (Toploc (Memory st)))" by simp
      show "¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) " using a2 by simp
    qed
    have b8:"∀ locs tp. CompMemType m len subT tp l'' locs ⟶ locs ≠ (ShowL_nat (Toploc (Memory st)))"
    proof intros
      fix locs tp
      assume asm1:"CompMemType m len subT tp l'' locs"
      then show "locs ≠ (ShowL_nat (Toploc (Memory st)))" using t7 bb a32
      proof(induction subT arbitrary:len l'')
        case (MTArray x1 subT)
        obtain i' ptr' where i'Def:"i'<len ∧ accessStore (hash l'' (ShowL_nat i')) m = Some (MPointer
ptr')
            ∧ (ptr' = locs ∧ MTArray x1 subT = tp ∨ CompMemType m x1 subT tp ptr' locs)"
          using MTArray.prems(1) unfolding CompMemType.simps by blast
        have "¬ TypedMemSubPref (hash l'' (ShowL_nat i')) (ShowL_nat (Toploc (Memory st))) (MTArray x
memArr)"

          by (metis MTArray.prems(2) bot_nat_0.extremum_unique i'Def leI limitSt mcon_accessStore
nat_less_le neg_MemLSubPrefL2_imps_TypedMemSubPref not_Some_eq)
        then have "accessStore (hash l'' (ShowL_nat i')) (Memory st) = accessStore (hash l'' (ShowL_nat
i')) m" using MTArray.prems memArrDef
          by (metis MConArrayPointers allocateSame bot_nat_0.extremum_strict bot_nat_0.not_eq_extremum
i'Def le_refl limitSt nonLocChanged option.distinct(1))
        then show ?case
        proof(cases "ptr' = locs")
          case True
          then show ?thesis
            using MTArray.prems(3) ‹accessStore (hash l'' (ShowL_nat i')) (Memory st) = accessStore (hash
l'' (ShowL_nat i')) m› i'Def by auto
        next
          case False
          then have "CompMemType m x1 subT tp ptr' locs"
            using i'Def by blast
          moreover have "MCon (MTArray x1 subT) (Memory st) ptr'"
            by (metis MCon_imps_sub_Mcon MTArray.prems(2) ‹accessStore (hash
                l'' (ShowL_nat i')) (Memory st) = accessStore (hash l'' (ShowL_nat i')) m›
                i'Def)
          moreover have "∀ locs tp. CompMemType (Memory st) x1 subT tp ptr' locs ⟶ locs ≠
              (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc
              (Memory st)))"
            using MTArray.prems(3)
            by (metis (no_types, lifting) CompTypeRemainsMCon MCon_imps_Some LSubPrefL2_def ‹MCon
(MTArray x1 subT) (Memory st)
                ptr'› Not_Sub_More_Specific le_refl limitSt option.distinct(1))
          ultimately show ?thesis using MTArray.IH[of x1 ptr'] using MTArray.prems by blast
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
    qed
```

```
    have nonLocChanged22:" ∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
        (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (Memory st) = accessStore locs m" using
        nonLocChanged
      by (simp add: allocateSame)
    then have l''mMcon:" MCon (MTArray len subT) m l''" using t7 bb
    proof(induction subT arbitrary: len l'')
      case (MTArray x1 subT)
      have "∀i<len.
           (case accessStore (hash l'' (ShowL_nat i)) m of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x1 subT of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x1 subT) m (hash l'' (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTArray x1 subT of MTArray len' arr' ⇒ MCon (MTArray x1
subT) m loc2 | MTValue Types ⇒ False))"
      proof(intros)
        fix i assume asm1:"i<len"
        then obtain ptr' where ptr'def:"accessStore (hash l'' (ShowL_nat i)) (Memory st) = Some
(MPointer ptr')
                        ∧ MCon (MTArray x1 subT) (Memory st) ptr'" using MTArray(3)
          by (metis MConArrayPointers MCon_imps_sub_Mcon MCon.simps(2))
        have "CompMemType (Memory st) len (MTArray x1 subT) (MTArray x1 subT) l'' (ptr')"
          using ptr'def asm1 by auto
        then have "ptr' ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 ptr' (ShowL_nat (Toploc
(Memory st)))"
          using MTArray.prems(3) by blast
        have same:"accessStore (hash l'' (ShowL_nat i)) (Memory st) = accessStore (hash l'' (ShowL_nat
i)) (m)" using nonLocChanged22
          by (metis Read_Show_nat'_id antisym_conv1 limitSt option.distinct(1) ptr'def
readLintNotEqual)

        have "∀locs tp. CompMemType (Memory st) x1 subT tp ptr' locs ⟶ locs ≠
            (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 locs (ShowL_nat (Toploc
            (Memory st)))"
          using MTArray.prems(3) <CompMemType (Memory st) len (MTArray x1 subT) (MTArray x1 subT) l''
ptr'> compMemTypes_trns by blast
        then have "MCon (MTArray x1 subT) m ptr'" using MTArray.IH[of x1 ptr'] ptr'def
          using nonLocChanged22 by blast
        then show "(case accessStore (hash l'' (ShowL_nat i)) m of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x1 subT of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x1 subT) m (hash l'' (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTArray x1 subT of MTArray len' arr' ⇒ MCon (MTArray x1
subT) m loc2 | MTValue Types ⇒ False))" using same ptr'def
          by (metis mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5))
      qed
      moreover have "len > 0" using MTArray.prems unfolding MCon.simps by blast
      moreover have "(∃p. accessStore l'' m = Some (MPointer p)) ∨ accessStore l'' m = None"
      proof(cases "l'' = l")
        case True
        then show ?thesis
          using lOrigin l_not_toploc_orSub nonLocChanged22 by auto
      next
        case False
        then have "accessStore (l'' ) (Memory st) = accessStore (l'') (m)" using nonLocChanged22
          by (metis MConArrayPointers MTArray.prems(2) Read_Show_nat'_id calculation(2)
lessThanSome_imps_Locs_hash lessThanTopLocs_def limitSt limitSt1 readLintNotEqual)
        then show ?thesis
          using MTArray.prems(2) calculation(2) by force
      qed
      ultimately show ?case using MCon.simps(2)[of len "MTArray x1 subT" m l'']
        by simp
    next
      case (MTValue x)
      have "∀i<len.
           (case accessStore (hash l'' (ShowL_nat i)) m of None ⇒ False
```

```
| Some (MValue val) ⇒ (case MTValue x of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTValue x) m
(hash l'' (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTValue x of MTArray len' arr' ⇒ MCon (MTValue x) m loc2 |
MTValue Types ⇒ False))"
    proof(intros)
      fix i assume asm1:"i<len"
      then obtain val where ptr':"accessStore (hash l'' (ShowL_nat i)) (Memory st) = Some (MValue
val)
                    ∧ MCon (MTValue x) (Memory st) (hash l'' (ShowL_nat i))" using MTValue(2)
        by (metis MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val)
      then have " CompMemType (Memory st) len (MTValue x) (MTValue x) l'' (hash l'' (ShowL_nat i))"
using asm1 by auto
      then have "(hash
          l'' (ShowL_nat i)) ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2
              (hash l'' (ShowL_nat i)) (ShowL_nat (Toploc (Memory st)))"
        using MTValue by blast
      then have "accessStore (hash l'' (ShowL_nat i)) (Memory st) = accessStore (hash l'' (ShowL_nat
i)) (m)" using nonLocChanged22 by auto
      then show "(case accessStore (hash l'' (ShowL_nat i)) m of None ⇒ False | Some (MValue val) ⇒
(case MTValue x of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTValue x) m (hash l'' (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTValue x of MTArray len' arr' ⇒ MCon (MTValue x) m loc2 |
MTValue Types ⇒ False))"
        using ptr' asm1
        by (metis mtypes.simps(6) memoryvalue.simps(5) Option.option.simps(5) MCon.simps(1))
    qed
    moreover have "(∃p. accessStore l'' m = Some (MPointer p)) ∨ accessStore l'' m = None"
    proof(cases "l'' =l")
      case True
      then show ?thesis
        using lOrigin l_not_toploc_orSub nonLocChanged22 by auto
    next
      case False
      then have "accessStore (l'' ) (Memory st) = accessStore (l'') (m)" using nonLocChanged22
        by (metis MCon_imps_Some MTValue.prems(2) Read_Show_nat'_id antisym_conv1
lessThanSome_imps_Locs_hash lessThanTopLocs_def limitSt limitSt1 option.discI readLintNotEqual)

      then show ?thesis
        by (metis MTValue.prems(2) MCon.simps(2))
    qed
    moreover have "len > 0" using MTValue by fastforce
    ultimately show ?case using MCon.simps(2)[of len "MTValue x" m l''] by simp
  qed

  have p''mMcon:"MCon (MTArray x memArr) m p''" using mconlOld
  proof(induction memArr arbitrary:x p'')
    case (MTArray x1 t)
    have samep'':"p'' ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬LSubPrefL2 p'' (ShowL_nat (Toploc (Memory
st)))"
      by (metis (no_types, lifting) MCon_imps_Some MTArray(2) LSubPrefL2_def Not_Sub_More_Specific
le_refl limitSt option.distinct(1))
    have "x > 0" using MTArray
      using zero_less_iff_neq_zero by force
    moreover have "(∃p. accessStore p'' m = Some (MPointer p)) ∨ accessStore p'' m = None" using
MTArray samep''
      using nonLocChanged22 calculation(1) by force
    moreover have "∀i<x. (case accessStore (hash p'' (ShowL_nat i)) m of None ⇒ False
              | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                  (MTArray x1 t) m (hash p'' (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray x1 t)
m loc2 | MTValue Types ⇒ False))"
    proof(intros)
      fix i assume asm1:"i<x"
      then obtain ptr where ptr':"accessStore (hash p'' (ShowL_nat i)) (Memory st) = Some (MPointer
ptr) ∧ MCon (MTArray x1 t) (Memory st) ptr" using MTArray
```

```
          by (meson MConArrayPointers MCon_imps_sub_Mcon <0 < x>)
        moreover have "accessStore (hash p'' (ShowL_nat i)) m = accessStore (hash p'' (ShowL_nat i))
(Memory st)"
          using MemLSubPrefL2_specific_imps_general
              <p'' ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 p''
              (ShowL_nat (Toploc (Memory st)))> nonLocChanged22 by metis
        moreover have "accessStore ptr (Memory st) = accessStore ptr (m)"
          using nonLocChanged22
          by (metis MCon_imps_Some Read_Show_nat'_id antisym_conv1 calculation(1)
lessThanSome_imps_Locs_hash lessThanTopLocs_def limitSt limitSt1 option.discI readLintNotEqual)
        moreover have "MCon (MTArray x1 t) m ptr" using MTArray.IH ptr' by blast
        ultimately show "case accessStore (hash p'' (ShowL_nat i)) m of None ⇒ False
      | Some (MValue val) ⇒ (case MTArray x1 t of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTArray
x1 t) m (hash p'' (ShowL_nat i)))
       | Some (MPointer loc2) ⇒ (case MTArray x1 t of MTArray len' arr' ⇒ MCon (MTArray x1 t) m loc2 |
MTValue Types ⇒ False)" by simp
      qed
      ultimately show ?case using MCon.simps(2)[of x "MTArray x1 t" m p''] by simp
    next
      case (MTValue x')
      have "p'' ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬LSubPrefL2 p'' (ShowL_nat (Toploc (Memory st)))"
        by (metis (no_types, lifting) MCon_imps_Some MTValue LSubPrefL2_def Not_Sub_More_Specific
le_refl limitSt option.distinct(1))
      then have samep':"accessStore p'' (Memory st) = accessStore p'' m"
        by (simp add: nonLocChanged22)
      have "∀i<x. (case accessStore (hash p'' (ShowL_nat i)) m of None ⇒ False
| Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon (MTValue x') m
(hash p'' (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') m
loc2 | MTValue Types ⇒ False))"
      proof(intros)
        fix i assume asm1:"i<x"
        then obtain val where "accessStore (hash
            p'' (ShowL_nat i)) (Memory st) = Some (MValue val) ∧ MCon (MTValue x') (Memory st) (hash p''
            (ShowL_nat i))" using MTValue
          by (meson MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val)
        moreover have "accessStore (hash p'' (ShowL_nat i)) (Memory st) = accessStore (hash p''
(ShowL_nat i)) (m)"
          using MemLSubPrefL2_specific_imps_general
              <p'' ≠ (ShowL_nat (Toploc (Memory st))) ∧ ¬ LSubPrefL2 p''
              (ShowL_nat (Toploc (Memory st)))> nonLocChanged22 by blast
        ultimately show "(case accessStore (hash p'' (ShowL_nat i)) m of None ⇒ False
              | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
(MTValue x') m (hash p'' (ShowL_nat i)))
              | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') m
loc2 | MTValue Types ⇒ False))"
          by auto
      qed
      moreover have "x > 0" using MTValue
        using zero_less_iff_neq_zero by force
      moreover have "(∃p. accessStore p'' m = Some (MPointer p)) ∨ accessStore p'' m = None" using
MTValue samep'
        using calculation(2) by auto
      ultimately show ?case using MCon.simps(2)[of x "MTValue x'" m p''] by simp
    qed

    have prntMconNew:"MCon (MTArray len' (MTArray x memArr)) (Memory st') prnt"
    proof -
      have " ∀i<len'.
           (case accessStore (hash prnt (ShowL_nat i)) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x memArr of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x memArr) (Memory st') (hash prnt (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTArray x memArr of MTArray len' arr' ⇒ MCon (MTArray x
memArr) (Memory st') loc2 | MTValue Types ⇒ False))"
```

```
      proof intros
        fix i' assume asm1:"i'<len'"
        then obtain ptr where ptrDef': "accessStore (hash prnt (ShowL_nat i')) (Memory st) =
Some(MPointer ptr)
                                  " and ptrDef'2:"MCon (MTArray x memArr) (Memory st) ptr"
          using mconPrnt by (metis MConArrayPointers MCon_imps_sub_Mcon bot_nat_0.not_eq_extremum
not_less_zero)

        show "case accessStore (hash prnt (ShowL_nat i')) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x memArr of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x memArr) (Memory st') (hash prnt (ShowL_nat i')))
            | Some (MPointer loc2) ⇒ (case MTArray x memArr of MTArray len' arr' ⇒ MCon (MTArray x
memArr) (Memory st') loc2 | MTValue Types ⇒ False)"
        proof(cases "i' = i")
          case True
          then show ?thesis
            using MCondest2 accessL ldef mInStd by force
        next
          case False
          then have "accessStore (hash prnt (ShowL_nat i')) (Memory st) = accessStore (hash prnt
(ShowL_nat i')) (Memory st')" using ldef
            by (metis hashesIntSame limitSt nle_le nonLocChanged2 option.distinct(1) ptrDef')
          then have same:"accessStore (hash prnt (ShowL_nat i')) (Memory st') = Some(MPointer ptr)"
using ptrDef' by simp

          have locRule:"∀ct locs ints. CompMemType (Memory st) len' (MTArray x memArr) ct prnt locs
⟶ locs
              ≠ prnt ∧ hash locs (ShowL_nat ints) ≠ l"
            using BothMConImpsNotCompMemType CompTypeRemainsMCon mconPrnt ldef ShowLNatDot
hash_injective by metis
          have "CompMemType (Memory st) len' (MTArray x memArr) (MTArray x memArr) prnt ptr" using
ptrDef' asm1 by auto
          then have "MCon (MTArray x memArr) (Memory st') ptr" using ptrDef'2 locRule
          proof(induction memArr arbitrary: x ptr len' prnt)
            case (MTArray x11 x12)
            have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
             | Some (MValue val) ⇒ (case MTArray x11 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                 (MTArray x11 x12) (Memory st') (hash ptr (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTArray x11 x12 of MTArray len' arr' ⇒ MCon (MTArray x11
x12) (Memory st') loc2 | MTValue Types ⇒ False))"
              proof intros
                fix i'' assume "i''<x"
                then obtain ptr' where ptr'Def:"accessStore (hash ptr (ShowL_nat i'')) (Memory st) =
Some(MPointer ptr')
                                                ∧ MCon (MTArray x11 x12) (Memory st) ptr'" using
MTArray.prems(2)
                  by (metis MConArrayPointers MCon_imps_sub_Mcon neq0_conv not_less_zero)
                have "ptr ≠ prnt" using MTArray.prems by blast
                then have "(hash ptr (ShowL_nat i'')) ≠ l" using MTArray.prems ShowLNatDot hash_injective
by blast
                then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st) = accessStore (hash ptr
(ShowL_nat i'')) (Memory st')"
                  by (metis less_or_eq_imp_le limitSt nonLocChanged2 option.discI ptr'Def)
                then have same2:"accessStore (hash ptr (ShowL_nat i'')) (Memory st') = Some(MPointer
ptr')" using ptr'Def by simp

                have "MCon (MTArray x11 x12) (Memory st) ptr'" using ptr'Def by blast
                moreover have "CompMemType (Memory st) x (MTArray x11 x12) (MTArray x11 x12) ptr ptr'"
using ptr'Def
                  using ‹i'' < x› by auto
                moreover have "∀ct locs ints. CompMemType (Memory st) x (MTArray x11 x12) ct ptr locs
⟶ locs
                    ≠ ptr ∧ hash locs (ShowL_nat ints) ≠ l"
                  by (metis BothMConImpsNotCompMemType CompTypeRemainsMCon MTArray.prems(1,2,3)
```

```
compMemTypes_trns)
            ultimately have "MCon (MTArray x11 x12) (Memory st') ptr'" using MTArray.IH[of x x11 ptr
ptr'] by blast
            then show "(case accessStore (hash ptr (ShowL_nat i'')) (Memory st') of None ⇒ False
           | Some (MValue val) ⇒ (case MTArray x11 x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
               (MTArray x11 x12) (Memory st') (hash ptr (ShowL_nat i'')))
           | Some (MPointer loc2) ⇒ (case MTArray x11 x12 of MTArray len' arr' ⇒ MCon (MTArray x11
x12) (Memory st') loc2 | MTValue Types ⇒ False))"
              using ptrDef' same2 by auto
          qed

          moreover have xNotZero:"x>0" using MTArray.prems(2)
            using bot_nat_0.not_eq_extremum by fastforce
          moreover have "(∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr
(Memory st') = None"
          proof(cases "ptr = l")
            case True
            then have " accessStore ptr (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))" using 15 by auto
            then show ?thesis by blast
          next
            case False
            then have pdef:"(∃p. accessStore ptr (Memory st) = Some (MPointer p)) ∨ accessStore ptr
(Memory st) = None"
              using MTArray.prems(2) MCon.simps(2)[of x _ "Memory st" ptr] xNotZero by simp
            then have "¬ LSubPrefL2 ptr (ShowL_nat (Toploc (Memory st)))" using limitSt1 limitSt
              by (metis (no_types, lifting) MCon_imps_Some MTArray.prems(2) LSubPrefL2_def
Not_Sub_More_Specific le_refl option.distinct(1))
            then have "accessStore ptr (Memory st) =accessStore ptr (Memory st') " using False
nonLocChanged2 by simp
            then show ?thesis using pdef by simp
          qed
          ultimately show ?case using MCon.simps(2)[of x "MTArray x11 x12" "Memory st'" ptr] by simp
        next
          case (MTValue x2)
          have "∀i<x. (case accessStore (hash ptr (ShowL_nat i)) (Memory st') of None ⇒ False
             | Some (MValue val) ⇒ (case MTValue x2 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                 (MTValue x2) (Memory st') (hash ptr (ShowL_nat i)))
             | Some (MPointer loc2) ⇒ (case MTValue x2 of MTArray len' arr' ⇒ MCon (MTValue x2)
(Memory st') loc2 | MTValue Types ⇒ False))"
          proof intros
            fix i'' assume "i''<x"
            then obtain val where oldDef:"accessStore (hash ptr (ShowL_nat i'')) (Memory st) =
Some(MValue val)
                                          ∧ MCon (MTValue x2) (Memory st) (hash ptr (ShowL_nat
i''))" using MTValue ptrDef'
              by (meson MCon_imps_sub_Mcon MCon_sub_MTVal_imps_val)
            have "(hash ptr (ShowL_nat i'')) ≠ l " using MTValue ldef ShowLNatDot hash_injective by
blast
            then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st) = accessStore (hash ptr
(ShowL_nat i'')) (Memory st')"
              by (metis less_or_eq_imp_le limitSt nonLocChanged2 option.discI oldDef)
            then have "accessStore (hash ptr (ShowL_nat i'')) (Memory st') = Some(MValue val)" using
oldDef by simp
            then show "case accessStore (hash ptr (ShowL_nat i'')) (Memory st') of None ⇒ False
                  | Some (MValue val) ⇒ (case MTValue x2 of MTArray l a ⇒ False | MTValue typ ⇒
                     MCon (MTValue x2) (Memory st') (hash ptr (ShowL_nat i'')))
                  | Some (MPointer loc2) ⇒ (case MTValue x2 of MTArray len' arr' ⇒ MCon (MTValue
x2) (Memory st') loc2 | MTValue Types ⇒ False)"
              using oldDef by auto
          qed
          moreover have xNotZero:"x>0" using MTValue(2)
            using bot_nat_0.not_eq_extremum by fastforce
          moreover have "(∃p. accessStore ptr (Memory st') = Some (MPointer p)) ∨ accessStore ptr
```

```
(Memory st') = None"
            proof(cases "ptr = 1")
              case True
              then have " accessStore ptr (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))" using 15 by auto
              then show ?thesis by blast
            next
              case False
              then have pdef:"(∃p. accessStore ptr (Memory st) = Some (MPointer p)) ∨ accessStore ptr
(Memory st) = None"
                using MTValue(2) MCon.simps(2)[of x _ "Memory st" ptr] xNotZero by simp
              then have "¬ LSubPrefL2 ptr (ShowL_nat (Toploc (Memory st)))" using limitSt1 limitSt
                by (metis (no_types, lifting) MCon_imps_Some MTValue(2) LSubPrefL2_def
Not_Sub_More_Specific le_refl option.distinct(1))
              then have "accessStore ptr (Memory st) =accessStore ptr (Memory st') " using False
nonLocChanged2 by simp
              then show ?thesis using pdef by simp
            qed

            ultimately show ?case using MCon.simps(2)[of x "MTValue x2" "Memory st'" ptr] by simp
          qed
          then show ?thesis using same asm1 by simp
        qed
      qed

      moreover have "len' > 0" using mconPrnt
        using ldef by auto
      moreover have "prnt ≠ 1" using ldef
        by (metis hash_inequality)
      moreover have "(∃p. accessStore prnt (Memory st') = Some (MPointer p)) ∨ accessStore prnt
(Memory st') = None"
      proof(cases "LSubPrefL2 prnt (ShowL_nat (Toploc (Memory st)))")
        case True
        then show ?thesis
          by (metis LSubPrefL2_def Not_Sub_More_Specific l_not_toploc_orSub ldef)
      next
        case False
        then show ?thesis using nonLocChanged2 calculation mconPrnt by simp
      qed
      ultimately show ?thesis using MCon.simps(2)[of len' "MTArray x memArr" "Memory st'" prnt]
mconPrnt nonLocChanged2 ldef by simp
    qed

    have compmemst':"∀locs lens subTs. CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt
                          ∧ MCon (MTArray lens subTs) (Memory st) locs
                         ⟶ CompMemType (Memory (st')) lens subTs (MTArray len' arr') locs prnt ∧
CompMemType (m) lens subTs (MTArray len' arr') locs prnt"
    proof intros
      fix locs lens subTs
      assume asm1:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt ∧ MCon (MTArray
lens subTs) (Memory st) locs"
      then have asm5:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt" by blast
      have asm6:"MCon (MTArray lens subTs) (Memory st) locs" using asm1 by blast
      have asm2:"∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ⟶ accessStore locs (Memory
(st)) = accessStore locs m"
        using nonLocChanged allocateSame by auto
      have mconl'':"MCon (MTArray len subT) (Memory st) l''" using lInfo(5) tParentType by simp

      have a5:"CompMemType m lens subTs (MTArray len' arr') locs prnt" using
cpm2mCompMemTypeOld_imps_CompMemType[OF asm5 asm2 ] asm6 limitSt
        by (metis (no_types, opaque_lifting) MCon_imps_Some LSubPrefL2_def eq_imp_le
Not_Sub_More_Specific not_None_eq)
      then have a10:"CompMemType (Memory st') lens subTs (MTArray len' arr') locs prnt ∧ CompMemType m
lens subTs (MTArray len' arr') locs prnt" using asm6
```

```
      proof(induction subTs arbitrary: lens locs)
        case (MTArray x11 x12)
        then have "CompMemType m lens (MTArray x11 x12) (MTArray len' arr') locs prnt" using a5 by
simp
        then obtain iIn lIn where iInDef:"iIn<lens ∧ accessStore (hash locs (ShowL_nat iIn)) m = Some
(MPointer lIn)
                  ∧ (lIn = prnt ∧ MTArray x11 x12 = MTArray len' arr' ∨ CompMemType m x11 x12 (MTArray
len' arr') lIn prnt)"
          unfolding CompMemType.simps by blast
        have same2:" accessStore (hash locs (ShowL_nat iIn)) m = accessStore (hash locs (ShowL_nat iIn))
(Memory st)"
          using MConArrayPointers MTArray.prems(2) allocateSame bot_nat_0.not_eq_extremum iInDef
le_eq_less_or_eq limitSt nonLocChanged not_less_zero option.discI by metis
        then have mcIn:"MCon (MTArray x11 x12) (Memory st) lIn" using same2 iInDef MTArray.prems
          by (metis MCon_imps_sub_Mcon)
        then show ?case
        proof(cases "lIn = prnt")
          case True
          have "prnt ≠ l" using ldef
            by (metis hash_inequality)
          moreover have "MTArray x11 x12 = MTArray len' arr'" using iInDef True
            by (metis CompMemType_imps_CompMemJustType MConSubTypes <MCon (MTArray x11 x12) (Memory st)
lIn> lsublocs mconPrnt CompMemJustType.simps(2))
          ultimately show ?thesis using True
            by (metis MConPtrsMustBeSubLocs NonChangeM'm iInDef lsublocs mInStd same2
CompMemType.simps(2))
        next
          case False
          then have "CompMemType m x11 x12 (MTArray len' arr') lIn prnt" using iInDef by blast
          then have cp:"CompMemType (Memory st') x11 x12 (MTArray len' arr') lIn prnt" using
MTArray.IH[of x11 lIn] mcIn by blast
          have "(hash locs (ShowL_nat iIn)) ≠ l"
            by (metis CompMemType_imps_CompMemJustType CompMemTypes_asc MConSubTypes
memoryvalue.inject(2) iInDef lOrigin lsublocs mcIn mconlOld option.inject same2
                CompMemJustType.simps(2))
          then have "accessStore (hash locs (ShowL_nat iIn)) m = accessStore (hash locs (ShowL_nat iIn))
(Memory st')"
            by (simp add: NonChangeM'm mInStd)
          then have "∃i<lens. ∃l. accessStore (hash locs (ShowL_nat i)) (Memory st') = Some (MPointer
l)
              ∧ (l = prnt ∧ MTArray x11 x12 = MTArray len' arr' ∨ CompMemType (Memory st') x11 x12
(MTArray len' arr') l prnt)"
            using False cp iInDef by metis
          then show ?thesis unfolding CompMemType.simps(2)
            using iInDef by blast
        qed
      next
        case (MTValue x2)
        then show ?case
          using compType by auto
      qed
      then show "CompMemType (Memory st') lens subTs (MTArray len' arr') locs prnt" by blast
      show "CompMemType m lens subTs (MTArray len' arr') locs prnt " using a10 by simp
    qed


    have l''MConst':"∀locs lens subTs. CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt
                        ∧ MCon (MTArray lens subTs) (Memory st) locs ∧ MCon (MTArray lens subTs) m
locs ⟶ MCon (MTArray lens subTs) (Memory st') locs"
    proof intros

      fix locs lens subTs
      assume asm6:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt ∧ MCon (MTArray
lens subTs) (Memory st) locs ∧ MCon (MTArray lens subTs) m locs "
```

```
      then have asm7:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt" by simp
      have asm8:"MCon (MTArray lens subTs) (Memory st) locs" using asm6 by blast
      have asm9:"MCon (MTArray lens subTs) m locs " using asm6 by blast
      show "MCon (MTArray lens subTs) (Memory st') locs "
      proof(cases "locs = prnt")
        case True
        then show ?thesis
          using MConPtrsMustBeSubLocs_CompMemType lsublocs lsublocs3 prntMconNew asm8 asm7 by blast
      next
        case False
        then have cpMemo:"CompMemType (Memory st) lens subTs (MTArray len' arr') locs prnt"
          using asm6 by blast
        then have asm10:"CompMemType m lens subTs (MTArray len' arr') locs prnt" using compmemst' asm8
by blast
        then have "CompMemType (Memory (st')) lens subTs (MTArray len' arr') locs prnt" using
compmemst'
          using asm8 cpMemo by blast
        have "CompMemType m lens subTs (MTArray x memArr) locs p''" using asm10 compType
          by (metis allocateSame compMemTypes_trns lOrigin l_not_toploc_orSub lsublocs nonLocChanged
CompMemType.simps(2))
        then have "∀ locs' loct. CompMemType m lens subTs loct locs locs' ∧ locs' = p'' ⟶ loct =
(MTArray x memArr)"
          using p''mMcon asm9 asm7
          using CompMemTypeSameLocsSameType by blast
        then show ?thesis using asm9 p''mMcon ldef lOrigin
        proof(induction subTs arbitrary: lens locs)
          case (MTArray x1 subT)
          have "∀ i<lens.
           (case accessStore (hash locs (ShowL_nat i)) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x1 subT of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x1 subT) (Memory st') (hash locs (ShowL_nat i)))
            | Some (MPointer loc2) ⇒ (case MTArray x1 subT of MTArray len' arr' ⇒ MCon (MTArray x1
subT) (Memory st') loc2 | MTValue Types ⇒ False))"
          proof intros
            fix i'' assume asm1:"i''<lens"
            then obtain ptr where ptrDef:"accessStore (hash locs (ShowL_nat i'')) m = Some (MPointer
ptr) ∧ MCon (MTArray x1 subT) m ptr" using MTArray.prems(2)
              by (metis MConArrayPointers MconSameTypeSameAccessWithTyping bot_nat_0.not_eq_extremum
less_nat_zero_code)

            show "(case accessStore (hash locs (ShowL_nat i'')) (Memory st') of None ⇒ False
            | Some (MValue val) ⇒ (case MTArray x1 subT of MTArray l a ⇒ False | MTValue typ ⇒ MCon
                (MTArray x1 subT) (Memory st') (hash locs (ShowL_nat i'')))
            | Some (MPointer loc2) ⇒ (case MTArray x1 subT of MTArray len' arr' ⇒ MCon (MTArray x1
subT) (Memory st') loc2 | MTValue Types ⇒ False))"
            proof(cases "ptr = p'')
              case True
              then have sameTp:" (MTArray x1 subT) = (MTArray x memArr)" using MTArray.prems ptrDef
asm1 by auto
              then show ?thesis
              proof(cases "(hash locs (ShowL_nat i'')) = l")
                case True
                then have "accessStore (hash locs (ShowL_nat i'')) (Memory st') = Some (MPointer
(ShowL_nat (Toploc (Memory st))))"
                  using accessL mInStd by auto
                then show ?thesis using sameTp MCondest2 by fastforce
              next
                case False
                then have "accessStore (hash locs (ShowL_nat i'')) m = accessStore (hash locs (ShowL_nat
i'')) (Memory st')"
                  by (simp add: NonChangeM'm mInStd)
                have "∀ locs loct. CompMemType m x1 subT loct ptr locs ∧ locs = p'' ⟶ loct = MTArray
x memArr"
                  using CompMemTypeSameLocsSameType ptrDef
```

```
                using MTArray.prems(1) asm1 CompMemType.simps(2) by blast
              moreover have "MCon (MTArray x1 subT) (Memory st') ptr" using MTArray.IH[of x1 ptr]
MTArray.prems ptrDef calculation by blast
              ultimately show ?thesis
                by (metis mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5) ‹accessStore
(hash locs
                    (ShowL_nat i'')) m = accessStore (hash locs (ShowL_nat i'')) (Memory
                    st')› ptrDef)
            qed
          next
            case False
            then have "(hash locs (ShowL_nat
                i'')) ≠ l" using MTArray.prems memoryvalue.inject(2) ‹accessStore (hash locs
(ShowL_nat
                    i'')) m = Some (MPointer ptr) ∧ MCon (MTArray x1 subT) m ptr› option.inject

              by (metis l_not_toploc_orSub nonLocChanged22)
            then have "accessStore (hash locs (ShowL_nat i'')) m = accessStore (hash locs (ShowL_nat
i'')) (Memory st')"
              by (simp add: NonChangeM'm mInStd)
            have "∀ locs loct. CompMemType m x1 subT loct ptr locs ∧ locs = p'' ⟶ loct = MTArray x
memArr"
              using CompMemTypeSameLocsSameType ptrDef
              using MTArray.prems(1) asm1 CompMemType.simps(2) by blast
            moreover have "MCon (MTArray x1 subT) (Memory st') ptr" using MTArray.IH[of x1 ptr]
MTArray.prems ptrDef calculation by blast
            ultimately show ?thesis
              by (metis mtypes.simps(5) memoryvalue.simps(6) Option.option.simps(5) ‹accessStore
(hash locs
                  (ShowL_nat i'')) m = accessStore (hash locs (ShowL_nat i'')) (Memory
                  st')› ptrDef)
          qed
        qed
        moreover have "lens > 0" using MTArray by force
        moreover have "(∃p. accessStore locs (Memory st') = Some (MPointer p)) ∨ accessStore locs
(Memory st') = None"
        proof(cases "locs = l")
          case True
          then show ?thesis
            using accessL mInStd by auto
        next
          case False
          then have "accessStore locs (Memory st') = accessStore locs m"
            by (simp add: NonChangeM'm mInStd)
          then show ?thesis using MTArray(3) calculation(2) by simp
        qed

        ultimately show ?case using MCon.simps(2)[of lens "MTArray x1 subT" "Memory st'" locs] by
simp
      next
        case (MTValue x')
        have "∀ i<lens.
         (case accessStore (hash locs (ShowL_nat i)) (Memory st') of None ⇒ False
          | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
              (MTValue x') (Memory st') (hash locs (ShowL_nat i)))
          | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') (Memory
st') loc2 | MTValue Types ⇒ False))"
        proof(intros)
          fix i'' assume asm1:"i''<lens"
          then obtain ptr where ptrDef:"accessStore (hash locs (ShowL_nat
              i'')) (m) = Some(MValue ptr) ∧ MCon (MTValue x') m (hash locs (ShowL_nat i''))"
            using MCon_sub_MTVal_imps_val MTValue.prems(2) MconSameTypeSameAccessWithTyping by
presburger
          then have "(hash locs (ShowL_nat i'')) ≠ l"
```

```
            using allocateSame lOrigin l_not_toploc_orSub nonLocChanged by fastforce
          then have "accessStore (hash locs (ShowL_nat i'')) (m) = accessStore (hash locs (ShowL_nat
i'')) (Memory st')"
            by (simp add: NonChangeM'm mInStd)
          then show "case accessStore (hash locs (ShowL_nat i'')) (Memory st') of None ⇒ False
          | Some (MValue val) ⇒ (case MTValue x' of MTArray l a ⇒ False | MTValue typ ⇒ MCon
              (MTValue x') (Memory st') (hash locs (ShowL_nat i'')))
          | Some (MPointer loc2) ⇒ (case MTValue x' of MTArray len' arr' ⇒ MCon (MTValue x') (Memory
st') loc2 | MTValue Types ⇒ False)"
            using ptrDef
            by (metis mtypes.simps(6) memoryvalue.simps(5) Option.option.simps(5) MCon.simps(1))
        qed
        moreover have "lens > 0" using MTValue by fastforce
        moreover have "(∃p. accessStore locs (Memory st') = Some (MPointer p)) ∨ accessStore locs
(Memory st') = None"
        proof(cases "locs = l")
          case True
          then show ?thesis
            using accessL mInStd by auto
        next
          case False
          then have "accessStore locs (Memory st') = accessStore locs m"
            by (simp add: NonChangeM'm mInStd)
          then show ?thesis using MTValue(2) calculation(2) by simp
        qed
        ultimately show ?case using MCon.simps(2)[of lens "MTValue x'" "Memory st'" locs] by simp
      qed
    qed
    have mcf:"∀x11 x12 x3. CompMemType (Memory st) x11 x12 (MTArray len subT) x3 l'' ∧ MCon (MTArray
x11 x12) (Memory st) x3
        ⟶ CompMemType m x11 x12 (MTArray len subT) x3 l''"
    proof(intros)
      fix x11 x12 x3
      assume asm3:"CompMemType (Memory st) x11 x12 (MTArray len subT) x3 l'' ∧ MCon (MTArray x11 x12)
(Memory st) x3"
      then show "CompMemType m x11 x12 (MTArray len subT) x3 l''"
      proof(induction x12 arbitrary: x11 x3)
        case (MTArray x1 x12)
        then obtain i'' ptr where ptrDef:"(i''<x11 ∧ accessStore (hash x3 (ShowL_nat i'')) (Memory st)
= Some
          (MPointer ptr) ∧ (ptr = l'' ∧ MTArray x1 x12 = MTArray len subT ∨ CompMemType (Memory st)
x1 x12 (MTArray len
          subT) ptr l''))"
          unfolding CompMemType.simps by blast
        then show ?case
        proof(cases "ptr = l''")
          case True
          then show ?thesis
            by (metis CompMemType.simps(2) MTArray.prems existingLocation_imps_allLocs_same limitSt
nle_le nonLocChanged22 option.discI
              ptrDef)
        next
          case False
          then have " CompMemType (Memory st) x1 x12 (MTArray len subT) ptr l''" using ptrDef by simp
          moreover have "MCon (MTArray x1 x12) (Memory st) ptr"
            using MCon_imps_sub_Mcon MTArray.prems ptrDef by blast
          ultimately show ?thesis using MTArray.IH[of x1 ptr]
            by (metis limitSt nat_le_linear nonLocChanged22 not_Some_eq ptrDef CompMemType.simps(2))
        qed
      next
        case (MTValue x)
        then show ?case by simp
      qed
```

qed

have *BT3:"∀ locs tp x t. ¬LSubPrefL2 locs (ShowL$_{nat}$ (Toploc (Memory st))) ⟶*
¬CompMemType (Memory (st')) x t tp (ShowL$_{nat}$ (Toploc (Memory st))) locs
∧ ¬TypedMemSubPrefPtrs (Memory (st')) x t (ShowL$_{nat}$ (Toploc (Memory st))) locs"
using *CompMemType_imps_TypedMemSubPrefPtrs LSubPrefL2_def SubPtrs_top selfPoint2* **by** *blast*

have *mconL''New:"MCon (MTArray len subT) (Memory st') l''"* **using** *lsublocs3*
proof(*cases "prnt = l''"*)
case *True*
then have *notComp:"¬CompMemType (Memory (st)) len subT (MTArray len' arr') l'' prnt"*
using *lsublocs BothMConImpsNotCompMemType[of len subT "Memory st" l'' "(MTArray len' arr')"] lsublocs3*
using *BothMConImpsNotCompMemType t7* **by** *blast*
then have *sameTypes:"len = len' ∧ arr' = subT "* **using** *lsublocs3* **by** *simp*
then show *?thesis* **using** *prntMconNew True*
using *lsublocs* **by** *blast*
next
case *False*
then have *"CompMemType (Memory (st)) len subT (MTArray len' arr') l'' prnt"* **using** *lsublocs3* **by** *simp*
then have *a:"CompMemType (Memory (st')) len subT (MTArray len' (MTArray x memArr)) l'' prnt"* **using** *compmemst' lsublocs*
using *t7* **by** *blast*
then show *?thesis* **using** *False l''MConst' t7*
using *‹CompMemType (Memory st) len subT (MTArray len' arr') l'' prnt› l''mMcon* **by** *blast*
qed

have *NonChangeM'mT:"∀ locs. accessTypeStore locs m = accessTypeStore locs m'"*
using *15(5)* **unfolding** *accessStore_def updateStore_def accessTypeStore_def* **by** *auto*

have *sameTypeAccess:" ∀ locs. ¬ LSubPrefL2 locs (ShowL$_{nat}$ (Toploc (Memory st))) ∨ locs = (ShowL$_{nat}$ (Toploc (Memory st))) ⟶*
accessTypeStore locs (Memory st) = accessTypeStore locs m"
using *15(4)* **unfolding** *cps2m_def*
using *cps2mSingleChange_Typed[of p "(ShowL$_{nat}$ (Toploc (Memory st)))" t "(Storage st (Address env))" " (snd (allocate (Memory st)))" x m]*
allocateTypeSameAccess **by** *metis*

have *sameTypeAccess2:" ∀ locs. ¬ TypedMemSubPref locs (ShowL$_{nat}$ (Toploc (Memory st))) (MTArray x memArr) ∨ locs = (ShowL$_{nat}$ (Toploc (Memory st))) ⟶*
accessTypeStore locs (Memory st) = accessTypeStore locs m"
using *15(4)* **unfolding** *cps2m_def*
using *cps2mSingleChange_typedPref_typed[of p "(ShowL$_{nat}$ (Toploc (Memory st)))" t "(Storage st (Address env))" " (snd (allocate (Memory st)))" x m]*
allocateTypeSameAccess memArrDef compatible_TypedStoSubpref_imps_TypedMemSubPref[OF 15(3)]
by *(metis sameTypeAccess)*

have *tps:"∀ destl'.*
TypedMemSubPref destl' (ShowL$_{nat}$ (Toploc (Memory st))) (MTArray x memArr) ⟶
(∃ stt. CompMemType m x memArr stt (ShowL$_{nat}$ (Toploc (Memory st))) destl' ∧
(case stt of MTArray parent_len parent_arr ⇒ ∀ i<parent_len. accessTypeStore (hash destl' (ShowL$_{nat}$ i)) m = Some parent_arr
| MTValue pval ⇒ accessTypeStore destl' m = Some (MTValue pval)))"
using *15(3,4) memArrDef* **unfolding** *cps2m_def*
using *cps2m_TypeCompChange[of p "(ShowL$_{nat}$ (Toploc (Memory st)))" t "(Storage st (Address env))" "(snd (allocate (Memory st)))" x m]*
by *simp*

have *selfPointMem:"∀ l l'.*
TypedMemSubPref l (ShowL$_{nat}$ (Toploc (Memory st))) (MTArray x memArr) ∧ accessStore l m = Some

```
(MPointer l') ⟶ l' = l"
      using selfPoint compatible_TypedStoSubpref_imps_TypedMemSubPref[OF 15(3)] memArrDef
      using selfPoint_root by presburger
    have topIndexType:" ∀i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i)) m =
Some memArr" using memArrDef
      using "15"(3,4) cps2m_TypeCompChangeIndexs cps2m_def by auto
    have a32Mem:"∀locs.
     ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr) ⟶
     accessStore locs (Memory st) = accessStore locs m"
      using a32 compatible_TypedStoSubpref_imps_TypedMemSubPref[OF 15(3)] memArrDef
      using nonLocChanged22 by blast
    have subPrefM:"subPrefixStructuralConsistency (m)"
      using cpm2m_subPrefixPersist[OF _ _ _ sameTypeAccess _ sameTypeAccess2 tps MCondest selfPointMem
topIndexType nonLocChanged22 a32Mem]
        ‹accessStore (ShowL_nat (Toploc (Memory st))) (snd (allocate (Memory st))) = None› allocateSame
2(1)
      unfolding TypeSafe_def by simp

    have nonLocChanged_TypedSafe:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                      ⟶ accessTypeStore loc (Memory st') = accessTypeStore loc
(Memory st)"
    proof -
      have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
          (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st))) =
          accessTypeStore locs m"
        using 15(4,5,6) cps2mSingleChange_Typed[of p "(ShowL_nat (Toploc (Memory st)))" t _ " (snd
(allocate (Memory st)))" x m ]
          cps2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t _ " (snd (allocate (Memory st)))" ] by
simp
      then have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
          (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st))) =
          accessTypeStore locs m'"
        using 15(6,5) unfolding accessTypeStore_def updateStore_def by force
      then show "∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore loc
(Memory st') = accessTypeStore loc (Memory st)"
        using allocateTypeSameAccess mInStd by metis
    qed

    have nonLocChanged_MemSafe:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ∧ loc ≠ l
                                    ⟶ accessStore loc (Memory st') = accessStore loc (Memory st)"
      using mInStd nonLocChanged allocateSame mInStd 15(5) unfolding updateStore_def accessStore_def
      by (metis accessStore_def nonLocChanged2)

    show ?thesis unfolding TypeSafe_def StateInvariant_def
    proof intros
      show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using 15 by simp
    next
      show "unique_locations (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have a0:" compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by blast
      then show "compPointers (Stack st') (Denvalue env)" using sameStack sameStorage by simp
    next
      show "safeContract (Accounts st') (Storage st')" using sameStorage 2(1) 15 unfolding
TypeSafe_def safeContract_def by auto
    next
      show "balanceTypes (Accounts st')" using 15 using 2(1) unfolding TypeSafe_def by simp
    next
      have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by simp
      then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
    next
      show "svalueTypes (Svalue env)" using 2(1) unfolding TypeSafe_def by simp
    next
      have *:"((∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶ accessStore
loc (Stack st) = None) ∧
```

```
          (∀ loc y. accessStore loc (Stack st) = Some y ⟶ (∃ tloc<Toploc (Stack st). LSubPrefL2 loc
(ShowL_nat tloc))) "
        using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
      have **:"Toploc (Stack st) = Toploc (Stack st')" using 15 unfolding updateStore_def by auto
      show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
      proof intros

        fix tloc loc
        assume h1:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have "Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)" using ** by simp
        then show "accessStore loc (Stack st') = None" using *
          by (simp add: sameStack)
      next
        fix loc y
        assume h1:" accessStore loc (Stack st') = Some y"
        then show "∃ tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)" using ** *
          by (metis sameStack)
      qed
    next
      show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def by simp
    next
      have a10:"Toploc (snd (allocate (Memory st))) = Toploc m"
        using cps2mTopLocSame[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))"
"(snd (allocate
            (Memory st)))" x m] 15(4) mInStd unfolding cps2m_def by fastforce
      then have a12:"Toploc m = Toploc m'" using 15(5) unfolding updateStore_def by auto
      have a15:"lessThanTopLocs (Memory st)" using 2 unfolding TypeSafe_def by simp
      have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding allocate_def by
simp
      show "lessThanTopLocs (Memory st')" unfolding lessThanTopLocs_def
      proof intros
        fix tloc loc
        assume b10: "Toploc (Memory st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
        then have b20:"Toploc (Memory st) ≤ tloc" using a10 tloc a12
          using mInStd by force

        then show "accessStore loc (Memory st') = None "
          by (metis (no_types, lifting) LSubPrefL2_def MemLSubPrefTransitive NonChangeM'm ‹∃ x.
accessStore l (Memory st) =
              Some x› a10 a12 allocateSame antisym_conv2 b10 hash_inequality
              hash_suffixes_associative hashesIntSame limitSt mInStd nonLocChanged option.discI
order_less_le_trans tloc)
      next
        fix loc y
        assume "accessStore loc (Memory st') = Some y "
        then show "∃ tloc<Toploc (Memory st'). LSubPrefL2 loc (ShowL_nat tloc)"
          by (metis NonChangeM'm ‹∃ x. accessStore l (Memory st) = Some x› a10 a12 allocateSameAccess
limitSt1 mInStd nonLocChanged order_less_trans tloc)
      qed
    next
      show "typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
        unfolding typeCompat_def
      proof intros
        fix tLook lLook
        assume inDen:" (tLook, lLook) |∈| fmran (Denvalue env)"
        show " denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd tLook lLook"
        proof(cases lLook)
          case (Stackloc x1)
          then obtain a where adef:"accessStore x1 (Stack st') = Some a" using inDen Stackloc 2(1)
unfolding TypeSafe_def typeCompat_def using sameStack nonChangedStack by fastforce
          then show ?thesis
          proof(cases "a")
            case (KValue x1)
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack nonChangedStack un-
```

```
folding TypeSafe_def typeCompat_def
              by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(17) )
          next
            case (KCDptr x2)
            then show ?thesis using Stackloc adef inDen Stackloc 2(1) sameStack nonChangedStack un-
folding TypeSafe_def typeCompat_def
              by (metis (no_types, lifting) denvalue.simps(5) Option.option.simps(5)
stackvalue.simps(18) )
          next
            case (KMemptr x3)
            then have "∃struct. tLook = type.Memory struct" using Stackloc adef inDen Stackloc 2(1)
KMemptr sameStack unfolding TypeSafe_def typeCompat_def by (cases tLook;force+)
            then obtain struct where structdef:"tLook = type.Memory struct" by blast

            then have mcOld:"MCon struct (Memory st) x3" using 2(1) unfolding TypeSafe_def
typeCompat_def
              using Stackloc adef inDen KMemptr sameStack by (auto split:type.splits denvalue.splits
stackvalue.splits option.splits)
            then have mcM:"MCon struct m x3" using PreExistMconNotChangeByToploc limitSt limitSt1
nonLocChanged22
              by auto
            have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              using 2(1) inDen KMemptr Stackloc structdef adef 15(6)
              unfolding TypeSafe_def denvalueTypeCorrectness_def by simp
            then have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some arr)
     | MTValue val ⇒ accessTypeStore x3 (Memory st) = Some (MTValue val))"
              by (metis (no_types, lifting))
            have nonLocChanged_TypedSafe:"∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st)))
                                        ⟶ accessTypeStore loc m = accessTypeStore loc (Memory st)"
            proof -
              have "∀locs. ¬ LSubPrefL2 locs (ShowL_nat (Toploc (Memory st))) ∨ locs =
                  (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore locs (snd (allocate (Memory st)))
                  = accessTypeStore locs m"
                using 15(4,5,6) cps2mSingleChange_Typed[of p "(ShowL_nat (Toploc (Memory st)))" t
"(Storage st (Address env))" " (snd (allocate (Memory st)))" x m ]
                  cps2m_def[of p "(ShowL_nat (Toploc (Memory st)))" x t "(Storage st (Address env))" "
(snd (allocate (Memory st)))" ] by simp
              then show "∀loc. ¬ LSubPrefL2 loc (ShowL_nat (Toploc (Memory st))) ⟶ accessTypeStore
loc m = accessTypeStore loc (Memory st)"
                using allocateTypeSameAccess mInStd by metis
            qed

            then have "accessTypeStore x3 (Memory st) = accessTypeStore x3 m"
              using limitSt limitSt1 mcOld
              by (metis AllPtrsNotTop_MCon lessThanTopLocs_def nat_le_linear)
            moreover have eq2:"∀i::nat. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) =
accessTypeStore (hash x3 (ShowL_nat i)) (m)"
              using nonLocChanged_TypedSafe limitSt limitSt1 mcOld AllPtrsNotTop_MCon
lessThanTopLocs_def
              by (metis MemLSubPrefL2_specific_imps_general nat_le_linear)
            ultimately have structss:"(case struct of
     MTArray len arr ⇒
       (∀i<len. accessTypeStore (hash x3 (ShowL_nat i)) m = Some arr)
     | MTValue val ⇒ accessTypeStore x3 m = Some (MTValue val))" using structss by presburger

            have "accessTypeStore l (Memory st) = Some (MTArray x memArr)"
              using lInfo 15 nonLocChanged memArrDef by simp
            then have accl:"accessTypeStore l (m) = Some (MTArray x memArr)"
```

```
            using nonLocChanged_TypedSafe
            using l_not_toploc_orSub by auto
          have "MCon struct (Memory st') x3"
            using cpm2m_singleLChange[OF mcM structss _ _ _ MCondest2 subPrefM] 15(5) mInStd
              NonChangeM'm accessL accl memArrDef
            by (metis lOrigin l_not_toploc_orSub nonLocChanged22)

          then show ?thesis using structdef KMemptr Stackloc adef inDen Stackloc 2(1) unfolding
TypeSafe_def typeCompat_def by simp

        next
          case (KStoptr x4)
          then show ?thesis using Stackloc
              adef sameStorage inDen Stackloc KStoptr 2(1) nonChangedStack sameStack
            unfolding TypeSafe_def typeCompat_def
              apply(cases tLook) by fastforce+
        qed
        next
        case (Storeloc x2)
        then show ?thesis using sameStorage inDen 2(1) unfolding TypeSafe_def typeCompat_def by
(cases tLook; force)
       qed
      qed
      then have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd "
using sameStack by simp
      next
      have "Accounts st'= Accounts st" using 15 by auto
      then show "fullyInitialised env (Accounts st') (Stack st')" using 2(3) unfolding
fullyInitialised_def
          using sameStack by presburger
     next
      show "denvalueTypeCorrectness env (Stack st') (Memory st') "
          unfolding denvalueTypeCorrectness_def
      proof(intros)
        fix t2 l2 ptr_loc sub_loc
        assume *:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env) ∧ accessStore l2 (Stack st')
= Some (KMemptr ptr_loc)"

        show "case t2 of
       MTArray len arr ⇒
       (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        proof -
          have sameACC:"accessStore l2 (Stack st') = accessStore l2 (Stack st)"
              using nonChangedStack * 15(5,6) unfolding accessStore_def updateStore_def by simp
          then have mcOld:"MCon t2 (Memory st) ptr_loc" using * by (metis 2(1) sameMemTSafe)
          then have old:"(case t2 of
                        MTArray len arr ⇒
         (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
              using sameACC 2(1) * TypeSafe_def denvalueTypeCorrectness_def by fastforce

          have inDenvalue:"(type.Memory t2, Stackloc l2) |∈| fmran (Denvalue env)" using * by simp

          then show ?thesis
          proof(cases t2)
            case (MTArray x11 x12)
            then have mcOld:"MCon (MTArray x11 x12) (Memory st) ptr_loc" using mcOld by blast
            have old':"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12)"
                using old MTArray by simp

            have conc0:"(∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some x12)"
            proof -
              have o:"∀ i<x11. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some x12" us-
```

```
ing old' by blast
            then have "∀ i<x11. ∃ v. accessStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some v"
                using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by auto
            then have "∀ i<x11. ¬ LSubPrefL2 (hash ptr_loc (ShowL_nat i)) (ShowL_nat (Toploc (Memory
st)))"
                using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by fastforce
            then show ?thesis using nonLocChanged_TypedSafe o nonLocChanged_MemSafe old' by auto
          qed
          then show ?thesis using MTArray by auto
          next
          case (MTValue x2)
          have "¬ LSubPrefL2 ptr_loc (ShowL_nat (Toploc (Memory st)))"
              using typeSafe_noDenElementOverToploc_mem[OF 2(1) inDenvalue] * sameACC
              using LSubPrefL2_def MemLSubPrefTransitive by metis
          then have "accessTypeStore ptr_loc (Memory st') = accessTypeStore ptr_loc (Memory st)"
              using "15"(6) allocateTypeSameAccess cpm2mSingleChange_Typed cpm2m_def mInStd
              using nonLocChanged_TypedSafe by blast
          then show ?thesis using MTValue old by simp
        qed
      qed
    qed
    next
    show "subPrefixStructuralConsistency (Memory st')"
        unfolding subPrefixStructuralConsistency_def
    proof intros
      fix locs tp
      assume in1:" accessTypeStore locs (Memory st') = Some tp "
      have sameTy:"∀ locs. accessTypeStore locs m = accessTypeStore locs (Memory st') "
          using mInStd by (metis NonChangeM'mT)
      then have in2:"accessTypeStore locs m = Some tp " using in1 by simp
      show "case accessStore locs (Memory st') of None ⇒ False
            | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st') locs ∧ tp = MTValue val ∧
accessTypeStore locs (Memory st') = Some tp
            | Some (MPointer p) ⇒
              ∃ len arr.
                 MCon tp (Memory st') p ∧
                 tp = MTArray len arr ∧
                 (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some arr)"
      proof(cases "locs = l")
        case True
        have "accessTypeStore l (Memory st) = Some (MTArray x memArr)"
            using lInfo 15 nonLocChanged memArrDef by auto
        then have accl:"accessTypeStore l (m) = Some (MTArray x memArr)"
            using nonLocChanged_TypedSafe l_not_toploc_orSub
            using sameTypeAccess by auto
        then have tpDef:"tp = MTArray x memArr"
            using sameTy True in1 by simp
        then have ptr:"accessStore locs (Memory st') = Some (MPointer (ShowL_nat (Toploc (Memory
st))))"
            using mInStd True by (metis accessL)
        have sameAccm:"accessStore l m = accessStore l (Memory st)"
            using l_not_toploc_orSub nonLocChanged22 by fastforce
        moreover have " MCon tp (Memory st') (ShowL_nat (Toploc (Memory st)))"
            using MCondest2 tpDef by blast
        moreover have "(∀ i<x. accessTypeStore (hash (ShowL_nat (Toploc (Memory st))) (ShowL_nat i))
(Memory st') = Some memArr)"
            using tpDef ptr subPrefM sameAccm accl unfolding subPrefixStructuralConsistency_def
            using sameTy topIndexType by presburger
        moreover have "
       (∀ v. accessStore (ShowL_nat (Toploc (Memory st))) (Memory st') = Some (MPointer v)
            ⟶ accessTypeStore (ShowL_nat (Toploc (Memory st))) (Memory st') = Some (MTArray x
memArr))"
            using LSubPrefL2_def selfPoint2 by blast
        ultimately show ?thesis using True tpDef ptr by auto
```

```
          next
          case False
          then have sameACC:"accessStore locs m = accessStore locs (Memory st')"
              using mInStd NonChangeM'm by simp
          then have in3:"(case accessStore locs m of None ⇒ False | Some (MValue v) ⇒ ∃val. MCon tp
m locs
              ∧ tp = MTValue val ∧ accessTypeStore locs m = Some tp
      | Some (MPointer p) ⇒
          ∃len arr.
             MCon tp m p ∧
             tp = MTArray len arr ∧
             (∀i<len. accessTypeStore (hash p (ShowL_nat i)) m = Some arr) )"
              using subPrefM in2 unfolding subPrefixStructuralConsistency_def by blast
          then show ?thesis
          proof(cases "accessStore locs m")
            case None
            then show ?thesis using in3 sameACC by simp
            next
            case (Some a)
            then show ?thesis
            proof(cases a)
              case (MValue x1)
              then obtain val where " MCon tp m locs ∧ tp = MTValue val ∧ accessTypeStore locs m =
Some tp"
                  using in3 Some by auto
              moreover have "MCon tp (Memory st') locs" using sameACC calculation by auto
              ultimately show ?thesis using sameACC Some MValue sameTy
                  by (auto split:option.splits)
              next
              case (MPointer x2)
              have "accessTypeStore l (Memory st) = Some (MTArray x memArr)"
                  using lInfo 15 nonLocChanged memArrDef by auto
              then have accl:"accessTypeStore l (m) = Some (MTArray x memArr)"
                  using nonLocChanged_TypedSafe l_not_toploc_orSub
                  using sameTypeAccess by auto
              then obtain len arr where s:"MCon tp m x2 ∧ tp = MTArray len arr ∧
                        (∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) m = Some arr)"
                  using Some in3 MPointer by auto
              then have mcM:"MCon tp m x2" by auto
              have lAcc:"updateStore l (MPointer (ShowL_nat (Toploc (Memory st)))) m = Memory st'"
                  using 15(5) mInStd by blast
              have "MCon tp (Memory st') x2"
                  using cpm2m_singleLChange[OF mcM _ lAcc _ accl MCondest2 subPrefM]
                    mInStd NonChangeM'm accessL lOrigin l_not_toploc_orSub nonLocChanged22 s
                  by auto

              moreover have "(∀i<len. accessTypeStore (hash x2 (ShowL_nat i)) (Memory st') = Some arr)"
                  using s sameTy by simp

              ultimately show ?thesis using Some MPointer sameACC sameTy s
                  by (auto split:option.splits)
            qed
          qed
        qed
      qed
      next
      have old:"(∀locs. (∃t. accessStore locs (Memory st) = Some t) = (∃tt. accessTypeStore locs
(Memory st) = Some tt))"
          using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
      have somesome:"∀destl'. TypedMemSubPref destl' (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)
           ⟶ (∃t. accessStore destl' m = Some t) = (∃tt. accessTypeStore destl' m = Some tt)"
          using 15(3,4) unfolding cps2m_def using cps2m_TypeCompChange_somesome[of p "(ShowL_nat (Toploc
(Memory st)))" t _ "(snd (allocate (Memory st)))" x m]
          using memArrDef by blast
```

```
      have a30T:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)
                ⟶ accessTypeStore locs (Memory st) = accessTypeStore locs m"
          using 15(3,4) unfolding cps2m_def using cps2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t _ " (snd (allocate (Memory st)))" x m ]
            mInStd allocateTypeSameAccess memArrDef
          by (metis (no_types, lifting) ext compatible_TypedStoSubpref_imps_TypedMemSubPref_neg
cps2mSingleChange_Typed)
      have lIs:"∃ x. accessStore l (Memory st) = Some x" using lInfo by (auto split:option.splits)
      show "SomeValSomeTyp (Memory st')"unfolding SomeValSomeTyp_def
      proof intros
        fix locs
        show "(∃ t. accessStore locs (Memory st') = Some t) = (∃ tt. accessTypeStore locs (Memory st') =
Some tt) "
        proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)")
          case True
          then have ss:"(∃ t. accessStore locs m = Some t) = (∃ tt. accessTypeStore locs m = Some tt)"
              using somesome by simp
          then show ?thesis
          proof(cases "locs = l")
            case True
            then show ?thesis using lInfo NonChangeM'mT accessL mInStd ss
                using lIs l_not_toploc_orSub nonLocChanged22 by auto
            next
            case False
            then show ?thesis using ss NonChangeM'mT NonChangeM'm mInStd by simp
          qed
          next
          case False
          then have acc1:"accessStore locs (Memory st) = accessStore locs m"
              by (metis False memArrDef a32Mem)
          have acc2:"accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
              using a30T False mInStd NonChangeM'mT by simp
          show ?thesis
          proof(cases "locs = l")
            case True
            then show ?thesis using lInfo acc2 accessL mInStd by force
            next
            case False
            then have "accessStore locs m = accessStore locs (Memory st')"
                using mInStd NonChangeM'm by auto
            then show ?thesis using old acc1 acc2 by metis
          qed
        qed
      qed
      next
      fix locs tt
      assume a1:" accessTypeStore locs (Memory st) = Some tt"
      then have some:"∃ v. accessStore locs (Memory st) = Some v"
          using 2(1) unfolding TypeSafe_def SomeValSomeTyp_def by blast
      have a30T:"∀ locs. ¬ TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)
                ⟶ accessTypeStore locs (Memory st) = accessTypeStore locs m"
          using 15(3,4) unfolding cps2m_def using cps2mSingleChange_typedPref_typed[of p "(ShowL_nat
(Toploc (Memory st)))" t _ " (snd (allocate (Memory st)))" x m ]
            mInStd allocateTypeSameAccess memArrDef
          by (metis (no_types, lifting) ext compatible_TypedStoSubpref_imps_TypedMemSubPref_neg
cps2mSingleChange_Typed)
      show " accessTypeStore locs (Memory st') = Some tt"
      proof(cases "TypedMemSubPref locs (ShowL_nat (Toploc (Memory st))) (MTArray x memArr)")
        case True
        then show ?thesis using some a1
            by (metis le_refl limitSt option.distinct(1) typedPrefix_imp_SubPref)
        next
        case False
        then have acc1:"accessStore locs (Memory st) = accessStore locs m"
```

```
          using a30 mInStd
          by (metis False memArrDef a32Mem)
      have acc2:"accessTypeStore locs (Memory st) = accessTypeStore locs (Memory st')"
          using a30T False mInStd NonChangeM'mT by simp
      show ?thesis
      proof(cases "locs = l")
        case True
        then show ?thesis using lInfo acc2 accessL mInStd a1 by argo
        next
        case False
        then show ?thesis using acc1 acc2 a1 by auto
      qed
    qed
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs
(Memory st') = Some (MPointer v')"
        by (metis accessL le_refl limitSt mInStd nonLocChanged2 option.distinct(1))
    next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      using NonChangeM'm lOrigin limitSt mInStd nonLocChanged22
      by (metis dual_order.refl memoryvalue.distinct(1) option.distinct(1) option.inject)
    next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
        (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using lOrigin limitSt nonLocChanged2
        by (metis (no_types, lifting) LSubPrefL2_def MemLSubPrefTransitive Read_Show_nat'_id
hash_flatten_right hash_inequality nat_less_le option.distinct(1))
    next
    have a10:"Toploc (snd (allocate (Memory st))) = Toploc m"
        using cps2mTopLocSame[of p "(ShowL_nat (Toploc (Memory st)))" t "(Storage st (Address env))"
"(snd (allocate (Memory st)))" x m]
          15(4) mInStd unfolding cps2m_def by fastforce
    moreover have a12:"Toploc m = Toploc m'" using 15(5) unfolding updateStore_def by auto
    moreover have tloc:"Toploc (Memory st) < Toploc (snd (allocate (Memory st)))" unfolding
allocate_def by simp
    ultimately show "Toploc (Memory st) ≤ Toploc (Memory st')"
        using mInStd by auto
  qed
 qed
qed
```

Assignment case 16: a storage mapping assigned to a storage pointer on the stack: the stack now points to the same mapping.

```
lemma TypeSafe_Assign_16:
 assumes "TypeSafe env (Accounts st) (Stack st) (Memory st) (Storage st) cd"
   and "stmt (ASSIGN lv ex) env cd st = Normal ((), st')"
   and "fullyInitialised env (Accounts st) (Stack st)"
   and "expr ex env cd (st(|state.Gas := state.Gas st - costs (ASSIGN lv ex) env cd st|)) (state.Gas st
- costs (ASSIGN lv
       ex) env cd st) = Normal ((KStoptr p, type.Storage (STMap t g')), g)"
   and "lexp lv env cd (st(|state.Gas := g|)) g = Normal((LStackloc l, type.Storage t'),t'')"
   and "t' = (STMap t g')"
   and "st' = st(|state.Gas := t'', Stack := updateStore l (KStoptr p) (Stack st)|)"
 shows "StateInvariant env st st' cd"
proof -
 let ?case = "StateInvariant env st st' cd"
 note 2 = assms(1-3)
 note 16 = assms(4-7)
 then have stackChanged:"∀l'. l' ≠ l ⟶ accessStore l' (Stack st) = accessStore l' (Stack st')"
unfolding updateStore_def accessStore_def by simp
 have t'InDen:"(type.Storage t', Stackloc l) |∈| fmran (Denvalue env)" using
lexpStackloc_imps_inDen[OF 16(2)] by blast
 then have lDen:"∀t'. (t', Stackloc l) |∈| fmran (Denvalue env) ⟶ t'= (type.Storage (STMap t g'))"
```

```
using 16 2(1) unfolding TypeSafe_def unique_locations_def by auto

  have pOrigin:"SCon (STMap t g') (extractValueType (KStoptr p)) (Storage (st⦇Gas := state.Gas st -
costs (ASSIGN lv ex) env cd st⦈)) (Address env)) ∧
      (∃xx. KStoptr p = KStoptr xx) ∧
      (∃stloc tp'' .
          (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue env) ∧
          (tp'' = (STMap t g') ∧ KStoptr p = KStoptr stloc ∨ extractValueType (KStoptr p) ≠ stloc ∧
CompStoType tp'' (STMap t g') stloc (extractValueType (KStoptr p))))"
    using 2(1) 16(1) 2(3) using
      exprTypeconInduct(3)[of ex env cd "(st⦇Gas := state.Gas st - costs (ASSIGN lv ex) env cd st⦈)"
"(state.Gas st - costs (ASSIGN lv ex) env cd st)" "KStoptr p" ]
    by (auto split:type.splits if_splits )
  obtain pParent pParentT where
    pOrigin:"SCon (STMap t g') (extractValueType (KStoptr p)) (Storage (st⦇Gas := state.Gas st - costs
(ASSIGN lv ex) env cd st⦈)) (Address env))
        ∧(type.Storage pParentT, Storeloc pParent) |∈| fmran (Denvalue env) ∧
          (pParentT = (STMap t g') ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p) ≠
pParent ∧
              CompStoType pParentT (STMap t g') pParent (extractValueType (KStoptr p)))"
    using pOrigin by auto
  have a0: "compPointers (Stack st) (Denvalue env)" using 2(1) unfolding TypeSafe_def by simp
  then have pParentRelations:"(∀tp2 l2 l2' st12.
      (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧

      (l2 = Stackloc l2' ∧ accessStore l2' (Stack (st)) = Some (KStoptr st12) ∨ l2 = Storeloc st12) ⟶
      (if TypedStoSubpref st12 pParent pParentT then CompStoType pParentT tp2 pParent st12 else if
TypedStoSubpref pParent st12 tp2 then CompStoType tp2 pParentT st12 pParent else True))"
    using compPointers_def pOrigin by blast
  have pSCon:"SCon (STMap t g') (extractValueType (KStoptr p)) (Storage (st) (Address env))" using
pOrigin by force

  — All locations that reach p have parent type MTArray x t, and likewise for all locations below p
  have pRelations:"(∀tp2 l2 l2' st12 .
  (type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧
  ((l2 = Stackloc l2' ∧ accessStore l2' (Stack st) = Some(KStoptr st12)) ∨ (l2 = Storeloc st12))
   ⟶
  (if TypedStoSubpref p st12 tp2 then CompStoType tp2 (STMap t g') st12 p
   else if TypedStoSubpref st12 p (STMap t g') then CompStoType (STMap t g') tp2 p st12
   else True))"
  proof intros
    fix tp2 l2 l2' st12
    assume in1:"(type.Storage tp2, l2) |∈| fmran (Denvalue env) ∧ (l2 = Stackloc l2' ∧ accessStore l2'
(Stack st) = Some (KStoptr st12) ∨ l2 = Storeloc st12)"

    show "if TypedStoSubpref p st12 tp2 then CompStoType tp2 (STMap t g') st12 p else if
TypedStoSubpref st12 p (STMap t g') then CompStoType (STMap t g') tp2 p st12 else True"
    proof(cases "l2 = Stackloc l2'")
      case StL2:True

      have "SCon pParentT pParent (Storage st (Address env))" using pOrigin using 2(1) unfolding
TypeSafe_def typeCompat_def by force
      have prnt:" pParentT = (STMap t g') ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p)
≠ pParent ∧
          CompStoType pParentT (STMap t g') pParent (extractValueType (KStoptr p))"
        using pOrigin by simp
      then have comp:"(if TypedStoSubpref st12 pParent pParentT then CompStoType pParentT tp2 pParent
st12
                  else if TypedStoSubpref pParent st12 tp2 then CompStoType tp2 pParentT st12 pParent
else True)"
        using pParentRelations in1 StL2 prnt by force
      then show ?thesis
      proof(cases "TypedStoSubpref p st12 tp2")
        case pToSt12:True
```

```
        then show ?thesis
        proof(cases "pParentT = (STMap t g')")
          case True
          then have cc1:"KStoptr p = KStoptr pParent " using prnt
            using comp_stotype_same_type_same_loc by blast
          then have cc2:"TypedStoSubpref pParent stl2 tp2" using pToStl2 by auto
          then have "CompStoType tp2 (STMap t g') stl2 p"
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case t2:True
            then show ?thesis using comp
              by (metis CompStoType_sameLoc_sameType stackvalue.inject(4) True cc1 cc2
typedStoSub_imps_negInv)
          next
            case False
            then have "CompStoType tp2 pParentT stl2 pParent" using comp
              by (simp add: cc2)
            then show ?thesis
              using True cc1 by auto
          qed
          then show ?thesis by (simp add: pToStl2)
        next
          case False
          then have cmp:" CompStoType pParentT (STMap t g') pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref by
simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            have "CompStoType tp2 (STMap t g') stl2 p " using pToStl2 prntStl2 CompStoType_sharedSub
cmp by simp
            then show ?thesis using pToStl2 by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STMap t g') stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
              then show ?thesis using pToStl2 by simp
            next
              case False
              then have "¬TypedStoSubpref p stl2 tp2" using f1 cmp tsp NotRelatedPrnt_imps_notRelatedChild[of
stl2 pParent pParentT tp2 p] by simp
              then show ?thesis using f1 pToStl2 cmp by simp
            qed
          qed
        qed
      next
        case f1:False
        then show ?thesis
        proof(cases " TypedStoSubpref stl2 p (STMap t g')")
          case stl2ToP:True
          have " CompStoType (STMap t g') tp2 p stl2"
          proof(cases "pParentT = (STMap t g')")
            case True
            then have cc1:"KStoptr p = KStoptr pParent " using prnt
              using comp_stotype_same_type_same_loc by blast
            then have cc2:"TypedStoSubpref stl2 pParent (STMap t g')" using stl2ToP by auto
            then have " CompStoType (STMap t g') tp2 p stl2"
            proof(cases "TypedStoSubpref stl2 pParent pParentT")
              case t2:True
              then have "CompStoType pParentT tp2 pParent stl2" using comp by simp
```

```
            then have "CompStoType (STMap t g') tp2 p stl2" using cc1 True by blast
            then show ?thesis by blast
          next
            case False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp False by simp
              then show ?thesis using True cc1 f1 by auto
            next
              case f2:False
              then show ?thesis using False True cc2 by blast
            qed
          qed
          then show ?thesis by blast
        next
          case False
          then have cmp:" CompStoType pParentT (STMap t g') pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref
by simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            then show ?thesis using CompStoType_sharedSub[OF prntStl2 stl2ToP cmp] by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STMap t g') stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
              then show ?thesis
                using CompStoType_sameLocNdTyp CompStoType_sharedSub stl2ToP by blast
            next
              case False
              then show ?thesis
                using NotReachablePrnt_imps_notReachableChild cmp f1 stl2ToP by blast
            qed
          qed
        qed
        then show ?thesis using f1 by (simp )
      next
        case False
        then show ?thesis
          using f1 by auto
      qed
    qed
  next
    case f3:False
    then have StL2:"l2 = Storeloc stl2"
      using in1 by auto

    then have "SCon pParentT pParent (Storage st (Address env))" using pOrigin using 2(1) unfolding
TypeSafe_def typeCompat_def by force
    have prnt:" pParentT = (STMap t g') ∧ KStoptr p = KStoptr pParent ∨ extractValueType (KStoptr p)
≠ pParent ∧
        CompStoType pParentT (STMap t g') pParent (extractValueType (KStoptr p))"
      using pOrigin by simp
    then have comp:"(if TypedStoSubpref stl2 pParent pParentT then CompStoType pParentT tp2 pParent
stl2
                  else if TypedStoSubpref pParent stl2 tp2 then CompStoType tp2 pParentT stl2 pParent
else True)"
```

```
        using pParentRelations in1 StL2 prnt by force
      then show ?thesis
      proof(cases "TypedStoSubpref p stl2 tp2")
        case pToStl2:True
        then show ?thesis
        proof(cases "pParentT = (STMap t g')")
          case True
          then have cc1:"KStoptr p = KStoptr pParent " using prnt
            using comp_stotype_same_type_same_loc by blast
          then have cc2:"TypedStoSubpref pParent stl2 tp2" using pToStl2 by auto
          then have "CompStoType tp2 (STMap t g') stl2 p"
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case t2:True
            then show ?thesis using comp
              by (metis CompStoType_sameLoc_sameType stackvalue.inject(4) True cc1 cc2
typedStoSub_imps_negInv)
          next
            case False
            then have "CompStoType tp2 pParentT stl2 pParent" using comp
              by (simp add: cc2)
            then show ?thesis
              using True cc1 by auto
          qed
          then show ?thesis by (simp add: pToStl2)
        next
          case False
          then have cmp:" CompStoType pParentT (STMap t g') pParent p" using prnt by simp
          then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref by
simp
          then show ?thesis
          proof(cases "TypedStoSubpref stl2 pParent pParentT")
            case True
            then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
            have "CompStoType tp2 (STMap t g') stl2 p " using pToStl2 prntStl2 CompStoType_sharedSub
cmp by simp
            then show ?thesis using pToStl2 by simp
          next
            case f1:False
            then show ?thesis
            proof(cases "TypedStoSubpref pParent stl2 tp2")
              case True
              then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
              then have "CompStoType tp2 (STMap t g') stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
              then show ?thesis using pToStl2 by simp
            next
              case False
              then have "¬TypedStoSubpref p stl2 tp2" using f1 cmp tsp NotRelatedPrnt_imps_notRelatedChild[of
stl2 pParent pParentT tp2 p] by simp
              then show ?thesis using f1 pToStl2 cmp by simp
            qed
          qed
        qed
      next
        case f1:False
        then show ?thesis
        proof(cases " TypedStoSubpref stl2 p (STMap t g')")
          case stl2ToP:True
          have " CompStoType (STMap t g') tp2 p stl2"
          proof(cases "pParentT = (STMap t g')")
            case True
            then have cc1:"KStoptr p = KStoptr pParent " using prnt
              using comp_stotype_same_type_same_loc by blast
            then have cc2:"TypedStoSubpref stl2 pParent (STMap t g')" using stl2ToP by auto
```

```
            then have " CompStoType (STMap t g') tp2 p stl2"
            proof(cases "TypedStoSubpref stl2 pParent pParentT")
              case t2:True
              then have "CompStoType pParentT tp2 pParent stl2" using comp by simp
              then have "CompStoType (STMap t g') tp2 p stl2" using cc1 True by blast
              then show ?thesis by blast
            next
              case False
              then show ?thesis
              proof(cases "TypedStoSubpref pParent stl2 tp2")
                case True
                then have "CompStoType tp2 pParentT stl2 pParent" using comp False by simp
                then show ?thesis using True cc1 f1 by auto
              next
                case f2:False
                then show ?thesis using False True cc2 by blast
              qed
            qed
            then show ?thesis by blast
          next
            case False
            then have cmp:" CompStoType pParentT (STMap t g') pParent p" using prnt by simp
            then have tsp:"TypedStoSubpref p pParent pParentT" using CompStoType_imps_TypedStoSubpref
by simp
            then show ?thesis
            proof(cases "TypedStoSubpref stl2 pParent pParentT")
              case True
              then have prntStl2:" CompStoType pParentT tp2 pParent stl2" using comp by simp
              then show ?thesis using CompStoType_sharedSub[OF prntStl2 stl2ToP cmp] by simp
            next
              case f1:False
              then show ?thesis
              proof(cases "TypedStoSubpref pParent stl2 tp2")
                case True
                then have "CompStoType tp2 pParentT stl2 pParent" using comp f1 by simp
                then have "CompStoType tp2 (STMap t g') stl2 p" using cmp CompStoType_trns[of tp2 ] by
blast
                then show ?thesis
                  using CompStoType_sameLocNdTyp CompStoType_sharedSub stl2ToP by blast
              next
                case False
                then show ?thesis
                  using NotReachablePrnt_imps_notReachableChild cmp f1 stl2ToP by blast
              qed
            qed
          qed
          then show ?thesis using f1 by simp
        next
          case False
          then show ?thesis using f1 by auto
        qed
      qed
    qed
  qed
  have storageSame:"(Storage st' (Address env)) = (Storage st (Address env))" using 16 by simp
  show ?thesis unfolding TypeSafe_def StateInvariant_def
  proof (intros)
    show tcN:"typeCompat (Denvalue env) (Stack st') (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st') (Memory st') (Storage st' (Address env)) cd t l'"
      proof (split denvalue.split, intros)
        fix loc assume a20:"l' = Stackloc loc"
```

```
        show "case accessStore loc (Stack st') of None ⇒ False
        | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
        | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
        | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
        | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
        proof (cases "loc = l")

          case False
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then have "accessStore loc (Stack st) = None" using 16(4) False by (simp
add:stackSingleUpdate)
            then show False using a30 a20 a10 assms TypeSafe_def typeCompat_def False 2(1)
              by (metis option.distinct(1) typeSafeLocExists)
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have a40:"accessStore loc (Stack st) = Some x2" using 16(4) False by (simp
add:stackSingleUpdate)
            then have a50:"(Memory st) = (Memory st')" using 16(4) by simp
            then have a60:"(Storage st) = (Storage st')" using 16(4) by simp
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False )
                | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
                | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _
⇒ False)
                | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
              using a10 a20 a30 a40 a50 a60 2(1) unfolding TypeSafe_def typeCompat_def by (cases x2;
cases t; force)
          qed
        next
          case sameLoc:True
          show ?thesis
          proof (split option.split, intros)
            assume a30:"accessStore loc (Stack st') = None"
            then show False using a20 a10 assms(1) sameLoc 16(4) notNoneUpdate by simp
          next
            fix x2 assume a30:"accessStore loc (Stack st') = Some x2"
            then have x2IsP:"x2 = KStoptr p" using 16 sameLoc by auto
            show "case x2 of KValue val ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
            | KCDptr stloc ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
            | KMemptr stloc ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
            | KStoptr stloc ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address env)) | _ ⇒ False)"
            proof(cases "x2")
              case (KValue x1)
              then show ?thesis using x2IsP by auto
            next
              case (KCDptr x2)
              then show ?thesis using x2IsP by (simp add:sameLoc)
            next
              case (KMemptr x3)
              then show ?thesis using x2IsP by (simp add:sameLoc)
            next
              case (KStoptr x4)
              then show ?thesis using x2IsP a10 a20 lDen storageSame sameLoc
                using pSCon by force
            qed
          qed
        qed
      next
```

```
        fix x2 assume a20:"l' = Storeloc x2"
        then have "(Storage st' (Address env)) = (Storage st (Address env))" using 16(4) by simp
        then show "case t of type.Storage typ ⇒ SCon typ x2 (Storage st' (Address env)) | _ ⇒ False"
using a10 a20 TypeSafe_def 2(1) pSCon a10 a20 lDen storageSame
        proof -
          have "⋀t d. (t, d) |∉| fmran (Denvalue env) ∨
                  (case d of Stackloc l ⇒ (case accessStore l (Stack st) of None ⇒ False
                  | Some (KValue l) ⇒ (case t of Value t ⇒ typeCon t l | _ ⇒ False )
                  | Some (KCDptr l) ⇒ (case t of Calldata m ⇒ MCon m cd l | _ ⇒ False )
                  | Some (KMemptr l) ⇒ (case t of type.Memory m ⇒ MCon m (Memory st) l | _ ⇒ False )
                  | Some (KStoptr l) ⇒ (case t of type.Storage s ⇒ SCon s l (Storage st (Address env))
| _ ⇒ False))
                  | Storeloc l ⇒ (case t of type.Storage s ⇒ SCon s l (Storage st (Address env)) | _
⇒ False))"
            using 2(1) unfolding TypeSafe_def typeCompat_def by auto
          then show ?thesis
            by (metis (no_types) denvalue.simps(6) a10 a20 storageSame)
        qed
      qed
    qed
    have "typeCompat (Denvalue env) (Stack st) (Memory st') (Storage st' (Address env)) cd"
      unfolding typeCompat_def
    proof intros
      fix t l' assume a10:"(t, l') |∈| fmran (Denvalue env)"
      show "denvalueCon (Stack st) (Memory st') (Storage st' (Address env)) cd t l'"
      proof(cases "l'")
        case (Stackloc x1)
        then show ?thesis
        proof(cases "x1 = l")
          case sameLoc:True
          then show ?thesis
          proof(cases "accessStore x1 (Stack st)")
            case None
            then show ?thesis using Stackloc a10 assms(1) sameLoc 16(4)
              by (metis 2(1) option.distinct(1) typeSafeAllStacklocsExist)
          next
            case (Some a)
            then obtain t''' where "t = type.Storage t'''" using lDen a10 Stackloc 2(1) sameLoc by
blast
            then have x2IsP:"∃p. a = KStoptr p" using 16 sameLoc 2(1) a10 Stackloc Some unfolding
TypeSafe_def typeCompat_def
              by (cases a, force,simp,fastforce+)
            then show ?thesis using x2IsP a10 Stackloc lDen storageSame sameLoc Some 2(1) unfolding
TypeSafe_def typeCompat_def
              using pSCon by force
          qed
        next
          case False
          then have "accessStore x1 (Stack st) = accessStore x1 (Stack st')"
            using stackChanged by auto
          then show ?thesis using a10 Stackloc tcN unfolding typeCompat_def by force
        qed
      next
        case (Storeloc x2)
        then show ?thesis using a10 tcN unfolding typeCompat_def by force
      qed
    qed
  next
    show "unique_locations (Denvalue env)" using 2(1) typeSafeUnique by auto
  next
    have "(Accounts st) = Accounts(st')" using 16 by simp
    then show "balanceTypes (Accounts st')" using balanceTypes_def balanceTypes_def 2(1)
typeSafeAccounts by simp
  next
```

```
    have a0:"compPointers (Stack st) (Denvalue env)" using 2(1) storageSame unfolding TypeSafe_def by
simp
    show " compPointers (Stack st') (Denvalue env)" unfolding compPointers_def
    proof(intros)
      fix tp1 tp2 l1 l22 l1' l2' stl1 stl2
      assume a1:"(type.Storage tp1, l1) |∈| fmran (Denvalue env) ∧
   (type.Storage tp2, l22) |∈| fmran (Denvalue env) ∧
   (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
   (l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l22 = Storeloc stl2)"
      have a2:"(∀x y. x |∈| fmran (Denvalue env) ∧ y |∈| fmran (Denvalue env) ∧ snd x = snd y ⟶ x =
y)"
        using 2(1) unfolding TypeSafe_def unique_locations_def by simp

      then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if
TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
      proof(cases "l1 = Storeloc stl1")
        case t1:True
        then show ?thesis
        proof(cases "l22 = Storeloc stl2")
          case True
          then show ?thesis using t1 a0 a1 unfolding compPointers_def by blast
        next
          case False
          then have a4:"l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2)" using
a1 by simp
          then show ?thesis
          proof(cases "l2' = l")
            case True
            then have "stl2 = p" using a4 16 by simp
            then show ?thesis using pRelations a1 a0 t1 True a4 lDen by blast
          next
            case False
            then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
            then show ?thesis using a0 a1 t1 unfolding compPointers_def by metis
          qed
        qed
      next
        case f1:False
        then have a4:"l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1)" using a1
by simp
        then show ?thesis
        proof(cases "l22 = Storeloc stl2")
          case t1:True
          then show ?thesis
          proof(cases "l1' = l")
            case True
            then have "stl1 = p" using a4 16 by simp
            then show ?thesis using pRelations a1 a0 t1 True a4 lDen
              by (metis CompStoType_sameLoc_sameType type.inject(4) typedStoSub_imps_negInv)
          next
            case False
            then have " accessStore l1' (Stack st') = accessStore l1' (Stack st)" using stackChanged
a4 by simp
            then show ?thesis using a0 a1 t1 unfolding compPointers_def by metis
          qed
        next
          case False
          then have a4':"l22 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2)" using
a1 by simp
          then show ?thesis
          proof(cases "l1' = l")
            case True
            then have p1:"stl1 = p" using a4 16 by simp
```

```
            then have tp1c:"tp1 = (STMap t g')" using lDen a1
              using True a4 by auto
            then show ?thesis
            proof(cases "l2' = l")
              case True
              then have p2:"st12 = p" using a4' 16 by auto
              then have tp2c:"tp2 = (STMap t g')" using lDen a1 True a4' by auto
              then show ?thesis using p2 p1 tp1c tp2c by auto
            next
              case False
              then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
              then show ?thesis
                by (metis CompStoType_sameLoc_sameType a1 p1 pRelations tp1c typedStoSub_imps_negInv)
            qed
          next
            case False
            then have same1:" accessStore l1' (Stack st') = accessStore l1' (Stack st)" using
stackChanged a4 by simp
            then show ?thesis
            proof(cases "l2' = l")
              case True
              then have p2:"st12 = p" using a4' 16 by auto
              then have tp2c:"tp2 = (STMap t g')" using lDen a1 True a4' by auto
              then show ?thesis using same1 pRelations p2 tp2c a1
                by metis
            next
              case False
              then have " accessStore l2' (Stack st') = accessStore l2' (Stack st)" using stackChanged
a4 by simp
              then show ?thesis using a1 a0 same1
                using compPointers_def by auto
            qed
          qed
        qed
      qed
    qed
  next
    show "svalueTypes (Svalue env)" using svalueTypes_def typeSafeSvalue 2(1) by simp
  next
    have "(Storage st') = (Storage st)" using 16(4) by simp
    then show "safeContract (Accounts st') (Storage st')" using 2(1) 16 unfolding safeContract_def
TypeSafe_def by auto
  next
    have a10:"Toploc (Stack st') = Toploc (Stack st)" using 16(4) unfolding updateStore_def by simp
    then have a20:"∃val. accessStore l (Stack st) = Some val" using t'InDen typeSafeLocExists 2(1)
TypeSafe_def by blast
    then have a30:"(∀tloc loc. Toploc (Stack st) ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                ∧(∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st).
LSubPrefL2 loc (ShowL_nat tloc)))"
      using 2(1) unfolding TypeSafe_def lessThanTopLocs_def by simp
    then have a40:"(∀tloc loc. Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc) ⟶
accessStore loc (Stack st) = None)
                ∧(∀loc y. accessStore loc (Stack st) = Some y ⟶ (∃tloc<Toploc (Stack st').
LSubPrefL2 loc (ShowL_nat tloc)))"
      using a10 by simp
    show "lessThanTopLocs (Stack st')" unfolding lessThanTopLocs_def
    proof intros
      fix tloc loc
      assume *:"Toploc (Stack st') ≤ tloc ∧ LSubPrefL2 loc (ShowL_nat tloc)"
      then show "accessStore loc (Stack st') = None"
      proof(cases "loc = l")
        case True
```

```
        then show ?thesis using * a10
          by (metis a20 a30 option.distinct(1))
      next
        case False
        then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 16(4) unfolding
updateStore_def accessStore_def by simp
        then show ?thesis using 2(1) a40 * a10 False a30 by simp
      qed
    next
      fix loc y
      assume *:" accessStore loc (Stack st') = Some y "
      show "∃tloc<Toploc (Stack st'). LSubPrefL2 loc (ShowL_nat tloc)"
      proof(cases "loc = l")
        case True
        then show ?thesis using *a10 a20 a30 by simp
      next
        case False
        then have a50:"accessStore loc (Stack st) = accessStore loc (Stack st')" using 16(4) unfolding
updateStore_def accessStore_def by simp
        then show ?thesis using * 2(1) a40 by simp
      qed
    qed
  next
    show "lessThanTopLocs cd" using 2(1) unfolding TypeSafe_def using 16 by auto
  next
    show "lessThanTopLocs (Memory st')" using 2(1) unfolding TypeSafe_def using 16 by auto
  next
    have "envAddressesWellFormed env" using 2(1) unfolding TypeSafe_def by auto
    then show "addressFormat (Address env)" and "addressFormat (Sender env)" by simp+
  next
    show "AddressTypes (Accounts st')" using 2(1) unfolding TypeSafe_def using 16 by simp
  next
    have accSame:"Accounts st'= Accounts st" using 16 by auto
    from 2(3) obtain c_fi ct_fi dud_fi where fi:"
         Type (Accounts st (Address env)) = Some (atype.Contract c_fi) ∧
         Contract env = c_fi ∧
         ep $$ c_fi = Some (ct_fi, dud_fi) ∧
         (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue env $$ id = Some (type.Storage v, Storeloc
id))) ∧
         (∀id v loc. Denvalue env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
         (∀t0 l0 p0.
             (type.Storage t0, Stackloc l0) |∈| fmran (Denvalue env) ∧ accessStore l0 (Stack st) = Some
(KStoptr p0) ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t0 l'
p0))"
      unfolding fullyInitialised_def by blast
    have fiPtrs':"∀t''' l''' p'''.
             (type.Storage t''', Stackloc l''') |∈| fmran (Denvalue env) ∧ accessStore l''' (Stack st')
= Some (KStoptr p''') ⟶
             (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t''' l'
p''')"
    proof (intro allI impI)
      fix t''' l''' p'''
      assume in1:"(type.Storage t''', Stackloc l''') |∈| fmran (Denvalue env)∧accessStore l''' (Stack
st') = Some (KStoptr p''') "
      show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue env) ∧ CompStoType t' t''' l'
p'''"
      proof(cases "l''' = l")
        case True
        then have "p''' = p" using 16 in1 unfolding accessStore_def updateStore_def by simp
        then show ?thesis using pOrigin
          by (metis CompStoType_sameLocNdTyp True type.inject(4) in1 lDen extractValueType.simps(4))
      next
        case False
```

```
        then have "accessStore l''' (Stack st') = accessStore l''' (Stack st)"
          using stackChanged by force
        then show ?thesis using fi in1
          by presburger
      qed
    qed
    show "fullyInitialised env (Accounts st') (Stack st')"
      unfolding fullyInitialised_def using fi fiPtrs' accSame by metis
  next
    have cc0:"∀ l ptr_loc. accessStore l (Stack st') = Some (KMemptr ptr_loc) ⟶ accessStore l (Stack
st) = Some (KMemptr ptr_loc)"
      using 16(4) unfolding updateStore_def accessStore_def by auto
    show "denvalueTypeCorrectness env (Stack st') (Memory st')"

      unfolding denvalueTypeCorrectness_def
    proof intros
      fix t l ptr_loc sub_loc
      assume "(type.Memory t, Stackloc l) |∈| fmran (Denvalue env) ∧
     accessStore l (Stack st') = Some (KMemptr ptr_loc)"
      then have "(case t of
       MTArray len arr ⇒
         (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st) = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
        using 2(1) unfolding TypeSafe_def denvalueTypeCorrectness_def using cc0 by blast
      moreover have "Memory st = Memory st'" using 16(4) by simp
      ultimately show "case t of
     MTArray len arr ⇒
       (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
     | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
        by metis
    qed
  next
    show "subPrefixStructuralConsistency (Memory st')" using 2(1) unfolding TypeSafe_def using 16(4)
by simp
  next
    show "SomeValSomeTyp (Memory st')" using 2(1) unfolding TypeSafe_def using 16(4) by simp
  next
    show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
      using 2(1) unfolding TypeSafe_def using 16(4) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MPointer v')"
      using 16(4) by simp
  next
    show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 16(4) by simp
  next
    show " ⋀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹ accessStore loc
       (Memory st) = None ⟹ accessStore loc (Memory st') = None"
      using 16(4) by simp
  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')" using 16(4) by simp
  qed
qed

end

end
```

17 Statement preservation (TypeSafe)

theory *TypeSafe*
  imports *TypeSafe_Assign*
begin

context *typesafe_base*
begin

Successful statement execution preserves the state invariant when the initial state is type-consistent and the environment is fully initialised. The assumptions of locale *typesafe_base* remain explicit prerequisites.

The proof is by induction over the statement semantics *local.stmt*, with one case per statement form. Each case shows that the resulting state is again type-safe, that the environment stays fully initialised, and that memory only grows (see *StateInvariant*).

- *SKIP* is immediate; *COMP*, *ITE* and *WHILE* follow from the induction hypotheses.
- The sixteen forms of *ASSIGN* are proved separately in *TypeSafe_Assign*.
- *INVOKE*, *EXTERNAL* and *NEW* run a body in a freshly initialised environment and use the initialisation results of *TypeSafe_Environment* with the induction hypothesis.
- *TRANSFER* uses the balance lemmas of *TypeSafe_Base_Types*.
- The two forms of *BLOCK* use the declaration results of *TypeSafe_Decl*.

theorem *TypeSafe_Statements:*
  assumes *"TypeSafe ev (Accounts st) (Stack st) (Memory st) (Storage st) cd"*
    and *normal:"stmt smt ev cd st = Normal((),st')"*
    and *inits:"fullyInitialised ev (Accounts st) (Stack st)"*
  shows *"StateInvariant ev st st' cd"*
  using *assms(1,2,3)*
proof *(induction arbitrary: st' rule:stmt.induct)*
  case *(1 e cd st)*
  have **:"st' = st⦇Gas := (state.Gas st) - costs SKIP e cd st⦈"* **using** *1 skip* **by** *simp*
  moreover have *"Stack st = Stack st'"* **using** *** **by** *simp*
  moreover have *memSame:"Memory st' = Memory st"* **using** *** **by** *simp*
  moreover have *"Storage st' = Storage st"* **using** *** **by** *simp*
  moreover have *"Accounts st' = Accounts st"* **using** *** **by** *simp*
  moreover have *"WrittenMem_between (Memory st) (Memory st') = {}"* **using** *calculation* **unfolding** *WrittenMem_between_def* **by** *simp*
  moreover have *"ReachableMem e (Stack st') (Memory st') = ReachableMem e (Stack st) (Memory st)"* **unfolding** *ReachableMem.simps* **using** *calculation* **by** *metis*

  ultimately show *?case* **using** *assms 1 ** **unfolding** *StateInvariant_def fullyInitialised_def* **by** *auto*
next
  case *(2 lv ex env cd st)*
  then show *?case*
  proof *(cases rule:assign[OF 2(2)])*
    case *(1 v t2 g l2 t' g' v')*
    from *2 1* **show** *?thesis* **by** *(rule TypeSafe_Assign_1)*
  next
    case *subcase2:(2 v t2 g locationChanged t' g' v')*
    from *2 subcase2* **show** *?thesis* **by** *(rule TypeSafe_Assign_2)*
  next
    case *(3 v t g l t' g' v')*
    from *2 3* **show** *?thesis* **by** *(rule TypeSafe_Assign_3)*
  next
    case *(4 p x t g l g' p' m t' st'')*
    from *2 4* **show** *?thesis* **by** *(rule TypeSafe_Assign_4)*
  next
    case *(5 p x t g l t' g' p' s)*
    from *2 5* **show** *?thesis* **by** *(rule TypeSafe_Assign_5)*
  next

```
    case (6 p x t g l t' g' s)
    from 2 6 show ?thesis by (rule TypeSafe_Assign_6)
  next
    case (7 p x t g l t' g' m m')
    from 2 7 show ?thesis by (rule TypeSafe_Assign_7)
  next
    case (8 p x t g l t' g')
    from 2 8 show ?thesis by (rule TypeSafe_Assign_8)
  next
    case (9 p x t g l t' g' p' s)
    from 2 9 show ?thesis by (rule TypeSafe_Assign_9)
  next
    case (10 p x t g l t' g' s)
    from 2 10 show ?thesis by (rule TypeSafe_Assign_10)
  next
    case (11 p x t g l t' g')
    from 2 11 show ?thesis by (rule TypeSafe_Assign_11)
  next
    case (12 p x t g l t' g' p' m st'')
    from 2 12 show ?thesis by (rule TypeSafe_Assign_12)
  next
    case (13 p x t g l t' g')
    from 2 13 show ?thesis by (rule TypeSafe_Assign_13)
  next
    case (14 p x t g l t' g' s)
    from 2 14 show ?thesis by (rule TypeSafe_Assign_14)
  next
    case (15 p x t g l t' g' m m')
    from 2 15 show ?thesis by (rule TypeSafe_Assign_15)
  next
    case (16 p t g' g l t'' t')
    from 2 16 show ?thesis by (rule TypeSafe_Assign_16)
  qed
next
  case (3 s1 s2 e cd st)
  show ?case
  proof(cases "state.Gas st > costs (COMP s1 s2) e cd st")
    case True
    then have a5:"assert Gas (λst. costs (COMP s1 s2) e cd st < state.Gas st) st = Normal ((), st)" by
simp
    then obtain sgas where *:"sgas = st⦇Gas := state.Gas st - costs (COMP s1 s2) e cd st⦈" by simp
    then have a6:"modify (λst. st⦇Gas := state.Gas st - costs (COMP s1 s2) e cd st⦈) st = Normal ((),
sgas)" by simp
    moreover have sStackSame:"Stack st = Stack sgas" using * by simp
    moreover have sgasSame:"Memory st = Memory sgas" using * by simp
    moreover have "Storage st = Storage sgas" using * by simp
    moreover have "Accounts st = Accounts sgas" using * by simp
    ultimately have a10:"TypeSafe e (Accounts sgas) (Stack sgas) (Memory sgas) (Storage sgas) cd"
      using 3(3) TypeSafe_def True by simp
    then obtain sts1 where s1:"stmt s1 e cd sgas = Normal ((), sts1)" using 3 * stmt.simps(3) comp by
meson
    have fi:"fullyInitialised e (Accounts sgas) (Stack sgas)" using 3(5) unfolding fullyInitialised_def
using * by simp
    then have cc0:"StateInvariant e sgas sts1 cd" using 3(1)[OF a5 a6 a10 s1] by blast

    then have ts: "TypeSafe e (Accounts sts1) (Stack sts1) (Memory sts1) (Storage sts1) cd"
      using a10 3(1)[of "()" st "()" sgas sts1] a5 a6 * s1 unfolding StateInvariant_def by simp
    then have "stmt s2 e cd sts1 = Normal ((),st')" using 3(4) * s1 using comp by fastforce
    then have cc2:"StateInvariant e sts1 st' cd" using 3(2)[of "()" st "()" sgas "()" "sts1" st'] a5
a6 * s1 ts fi
      using atype_same by (metis cc0 StateInvariant_def)
    then have cc5:"TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd∧
  (∀locs t. accessTypeStore locs (Memory sts1) = Some t ⟶ accessTypeStore locs (Memory st') = Some t)
∧(∀i loc. i < Toploc (Memory sts1) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶ accessStore loc
```

```
    (Memory sts1) = None ⟶ accessStore loc (Memory st') = None)"
      unfolding StateInvariant_def by blast
    have cc10:"fullyInitialised e (Accounts st') (Stack st')∧
  (∀locs t. accessTypeStore locs (Memory sts1) = Some t ⟶ accessTypeStore locs (Memory st') = Some t)
∧ (∀i loc. i < Toploc (Memory sts1) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶ accessStore loc
    (Memory sts1) = None ⟶ accessStore loc (Memory st') = None)"
      using cc2 unfolding StateInvariant_def by blast
    then have cc12:"TypeSafe e (Accounts sts1) (Stack sts1) (Memory sts1) (Storage sts1) cd ∧
    fullyInitialised e (Accounts sts1) (Stack sts1)∧
    (∀locs t. accessTypeStore locs (Memory sgas) = Some t ⟶ accessTypeStore locs (Memory sts1) = Some
t) ∧
(∀i loc. i < Toploc (Memory sgas) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶ accessStore loc (Memory
    sgas) = None ⟶ accessStore loc (Memory sts1) = None)"
      using cc0 unfolding StateInvariant_def by simp
    have cc15:"TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd ∧
  fullyInitialised e (Accounts st') (Stack st') ∧
  (∀locs t. accessTypeStore locs (Memory sts1) = Some t ⟶ accessTypeStore locs (Memory st') = Some t)
∧ (∀i loc. i < Toploc (Memory sts1) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶ accessStore loc
    (Memory sts1) = None ⟶ accessStore loc (Memory st') = None)"
      using cc2 unfolding StateInvariant_def by blast
    have tlocs:"Toploc (Memory st) ≤ Toploc (Memory sts1) ∧ Toploc (Memory sts1) ≤ Toploc (Memory
st')"
      using cc0 cc2 unfolding StateInvariant_def using sgasSame by simp
    then have "(∀i loc. i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i)
              ⟶ accessStore loc (Memory st) = None ⟶ accessStore loc (Memory st') = None)"
      using sgasSame cc15 cc12
      by (metis (no_types, lifting) dual_order.strict_trans1)
    then show ?thesis using cc5 cc10 cc15 cc12 unfolding StateInvariant_def
      using sgasSame
      using StateInvariant_def cc0 cc2 tlocs by auto
  next
    case False
    then have "assert Gas (λst. costs (COMP s1 s2) e cd st < state.Gas st) st = Exception Gas" by simp
    then show ?thesis using 3(4) stmt.simps(3) by simp
  qed
next
  case (4 ex s1 s2 e cd st)
  then show ?case
  proof(cases "state.Gas st> costs (ITE ex s1 s2) e cd st")
    case True
    then have a5:"assert Gas (λst. costs (ITE ex s1 s2) e cd st < state.Gas st) st = Normal ((), st)"
by simp
    then obtain sgas where *:"sgas = st⦇Gas := state.Gas st - costs (ITE ex s1 s2) e cd st⦈" by simp
    then have a6:"modify (λst. st⦇Gas := state.Gas st - costs (ITE ex s1 s2) e cd st⦈) st = Normal
((), sgas)" by simp
    moreover have "Stack st = Stack sgas" using * by simp
    moreover have "Memory st = Memory sgas" using * by simp
    moreover have "Storage st = Storage sgas" using * by simp
    moreover have "Accounts st = Accounts sgas" using * by simp
    ultimately have a10:"TypeSafe e (Accounts sgas) (Stack sgas) (Memory sgas) (Storage sgas) cd" using
4(3) True by simp
    then consider (atrue) g where "expr ex e cd sgas (state.Gas sgas) = Normal ((KValue (ShowL_bool
True), Value TBool),g)"
      |(afalse) g where "expr ex e cd sgas (state.Gas sgas) = Normal ((KValue (ShowL_bool False), Value
TBool),g)" using * 4(4) stmt.simps(4)
      by (simp split:if_split_asm result.split_asm prod.split_asm stackvalue.split_asm type.split_asm
types.split_asm)
    then show ?thesis
    proof (cases)
      case atrue
      then obtain g where a20:"expr ex e cd sgas (state.Gas sgas) = Normal ((KValue (ShowL_bool True),
Value TBool), g)" by simp
      then obtain s'' where a25:"s'' = sgas⦇Gas:= g⦈" by simp
      moreover have "Stack s'' = Stack sgas" using a25 by simp
```

```
      moreover have "Memory s'' = Memory sgas" using a25 by simp
      moreover have "Storage s'' = Storage sgas" using a25 by simp
      moreover have "Accounts s'' = Accounts sgas" using a25 by simp
      ultimately have "TypeSafe e (Accounts s'') (Stack s'') (Memory s'') (Storage s'') cd" using assms
4 True atrue a10 a25 by metis
      moreover have "stmt s1 e cd s'' = Normal((),st')" using * a20 stmt.simps(4) 4(4) a25 by (simp
split:if_split_asm result.split_asm prod.split_asm)
      moreover have "fullyInitialised e (Accounts s'') (Stack s'')" using 4(5)
        using ‹Accounts s'' = Accounts sgas› ‹Accounts st = Accounts sgas›
        by (simp add: ‹Stack s'' = Stack sgas› ‹Stack st = Stack sgas›)
      ultimately have "StateInvariant e s'' st' cd" using 4(1)[of "()" st "()" sgas "(KValue (ShowL_bool
True),
          Value TBool)" s'' "(ShowL_bool True)" s'' st'] a5 a6 a20 a25
        unfolding StateInvariant_def by simp
      then show ?thesis unfolding StateInvariant_def
        using ‹Memory s'' = Memory sgas› ‹Memory st = Memory sgas› by simp
    next
      case afalse
      then obtain g where a20:"expr ex e cd sgas (state.Gas sgas) = Normal ((KValue (ShowL_bool False),
Value TBool), g)" and
        a21: "expr ex e cd sgas (state.Gas sgas) ≠ Normal ((KValue (ShowL_bool True), Value TBool), g)"
by fastforce
      then obtain s'' where a25:"s'' = sgas(|Gas:= g|)" by simp
      moreover have "Stack s'' = Stack sgas" using a25 by simp
      moreover have "Memory s'' = Memory sgas" using a25 by simp
      moreover have "Storage s'' = Storage sgas" using a25 by simp
      moreover have "Accounts s'' = Accounts sgas" using a25 by simp
      ultimately have "TypeSafe e (Accounts s'') (Stack s'') (Memory s'') (Storage s'') cd" using assms
4 True afalse a10 a25 by metis
      moreover have "stmt s2 e cd s'' = Normal((),st')" using * a20 a21 stmt.simps(4)[of ex s1 s2 e cd
st ] 4(4) a25 by (simp split:if_split_asm result.split_asm prod.split_asm)
      moreover have "fullyInitialised e (Accounts s'') (Stack s'')" using 4(5) unfolding
fullyInitialised_def
        using ‹Accounts s'' = Accounts sgas› ‹Accounts st = Accounts sgas›
        by (simp add: ‹Stack s'' = Stack sgas› ‹Stack st = Stack sgas›)
      moreover have "StateInvariant e s'' st' cd" using 4(2)[of "()" st "()" sgas "(KValue (ShowL_bool
False),
          Value TBool)" s'' "(ShowL_bool False)" s'' st']
          a5 a6 a20 a21 a25 calculation by auto
      ultimately show ?thesis
        unfolding StateInvariant_def
        using ‹Memory s'' = Memory sgas› ‹Memory st = Memory sgas› by auto
    qed
  next
    case False
    then show ?thesis using 4(4) stmt.simps(4) by simp
  qed
next
  case (5 ex s0 e cd st)
  show ?case
  proof(cases rule:while[OF 5(4)])
    case (1 g st'')
    then have a5:"assert Gas (λst. costs (WHILE ex s0) e cd st < state.Gas st) st = Normal ((), st)"
by simp
    then obtain sgas where *:"sgas = st(|Gas := state.Gas st - costs (WHILE ex s0) e cd st|)" by simp
    then have a6:"modify (λst. st(|Gas := state.Gas st - costs (WHILE ex s0) e cd st|)) st = Normal ((),
sgas)" by simp

    then have a10:"toState (expr ex e cd) sgas
          = Normal ((KValue (ShowL_bool True), Value TBool), (st(|Gas := g|)))" using 1 * by auto


    have a20:"TypeSafe e (Accounts (st(|Gas := g|))) (Stack (st(|Gas := g|))) (Memory (st(|Gas := g|)))
(Storage (st(|Gas := g|))) cd"
```

```
      using 5(3) by simp
    have a25:"fullyInitialised e (Accounts (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈))" using 5(5) unfold-
ing fullyInitialised_def by simp
    then have a30:"StateInvariant e (st⦇Gas := g⦈) st'' cd"
      using 5(1)[OF a5 a6 a10 _ _ a20 1(3) ] unfolding StateInvariant_def by simp
    have cc0:"StateInvariant e st'' st' cd"
      using 5(2)[OF a5 a6 a10 _ _ 1(3) _ 1(4)] a30 unfolding StateInvariant_def by simp

    then have ts:"TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd ∧
  fullyInitialised e (Accounts st') (Stack st')" using cc0 unfolding StateInvariant_def by blast
    have "Memory st = Memory (st⦇Gas := g⦈) ∧ Stack st = Stack (st⦇Gas := g⦈)" by simp
    then show ?thesis using ts a30 cc0 unfolding StateInvariant_def by fastforce
  next
    case (2 g)
    then have "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd "
      by (simp add: "5.prems"(1))
    moreover have "fullyInitialised e (Accounts st') (Stack st')" using 2
      using "5.prems"(2,3) atype_same fullyInitialised_def by simp
    moreover have "Memory st = Memory st' ∧ Stack st = Stack st'" using 2 by simp
    ultimately show ?thesis unfolding StateInvariant_def by simp
  qed
next
  case (6 i xe e cd st)
  show ?case
  proof(cases rule:invoke[OF 6(3)])
    case (1 ct fb fp f e_l cd_l k_l m_l g st'')
    then have a5:"assert Gas (λst. costs (INVOKE i xe) e cd st < state.Gas st) st = Normal ((), st)"
by simp
    then obtain sgas where *:"sgas = st⦇Gas := state.Gas st - costs (INVOKE i xe) e cd st⦈" by simp
    then have a6:"modify (λst. st⦇Gas := state.Gas st - costs (INVOKE i xe) e cd st⦈) st = Normal ((),
sgas)" by simp
    have a10:"option Err (λ_. ep $$ Contract e) sgas = Normal ((ct, fb), sgas)" using 1(2) by simp
    then have a20:"(case ct $$ i of None ⇒ throw Err | Some (Method (fp, True, f)) ⇒ throw Err
     | Some (Method (fp, False, f)) ⇒ return (fp, f) | Some _ ⇒ throw Err)
     sgas = Normal ((fp, f), sgas)" using 1 by auto


    have memSGas:"Memory sgas = Memory st" using * by auto
    have a25:"applyf Memory sgas = Normal ((Memory sgas), sgas)" by auto

    obtain x where xDef:"x = ffold_init ct (emptyEnv (Address e) (Contract e) (Sender e) (Svalue e))
(fmdom ct)" by simp

    have tsX:"TypeSafe x (Accounts sgas) emptyStore (Memory sgas) (Storage sgas) emptyTypedStore"
      using ffoldInitTypeSafe[of "(Accounts sgas)" "(Storage sgas)" "Svalue e" "(Memory sgas)" _ ct fb
"Address e" _ _]
      using * 6(2) "6.prems"(3) unfolding TypeSafe_def fullyInitialised_def using 1(2) xDef
ffold_init_def by auto
    have cx:"Contract x = Contract e" using xDef by simp
    then have a27:"load False fp xe x emptyTypedStore emptyStore (Memory sgas) e cd (sgas) (state.Gas
st - costs (INVOKE i xe) e cd st) =
    Normal ((e_l, cd_l, k_l, m_l), g)" using 1(4) * xDef unfolding ffold_init_def by simp
    then have a30:"toState (load False fp xe x emptyTypedStore emptyStore (Memory sgas) e cd) sgas =
  Normal ((e_l, cd_l, k_l, m_l), sgas⦇Gas := g⦈) " using * 1(5) by auto
    have sameADD:"Address e = Address x"
      by (simp add: xDef)
    have link:"TypeSafe e_l (Accounts sgas) k_l m_l (Storage sgas) cd_l ∧
    fullyInitialised e_l (Accounts sgas) k_l ∧
    (¬ False ⟶
     (∀ locs tp. MCon tp (Memory sgas) locs ⟶ MCon tp m_l locs) ∧
     Toploc (Memory sgas) ≤ Toploc m_l ∧
     ncpDenvalueLimit e_l e k_l (Stack sgas) (Memory sgas) ∧
     ncpOMemInDMem (Memory sgas) m_l ∧ ncpElementsNoSubPref (Memory sgas) m_l ∧ ncpNewSelfPoint (Memory
sgas) m_l)"
```

```
    proof -
      have "(∀ locs tp. MCon tp (Memory sgas) locs ⟶ MCon tp (Memory sgas) locs)" by blast
      moreover have "Toploc (Memory sgas) ≤ Toploc (Memory sgas)" by simp
      moreover have "ncpDenvalueLimit x e emptyStore (Stack sgas) (Memory sgas)" unfolding
ncpDenvalueLimit_def accessStore_def emptyStore_def by simp
      moreover have "ncpOMemInDMem (Memory sgas) (Memory sgas)" unfolding ncpOMemInDMem_def by blast
      moreover have "ncpElementsNoSubPref (Memory sgas) (Memory sgas)" using
ncpElementsNoSubPref_sameMem 6(2) * by auto
      moreover have "ncpNewSelfPoint (Memory sgas) (Memory sgas)" unfolding ncpNewSelfPoint_def by
auto
      moreover have "TypeSafe x (Accounts sgas) emptyStore (Memory sgas) (Storage sgas)
emptyTypedStore"
        using ffoldInitTypeSafe[of "(Accounts sgas)" "(Storage sgas)" "Svalue e" "(Memory sgas)" e ct
fb "Address e" _ _]
        using * 6(2) "6.prems"(3) unfolding TypeSafe_def fullyInitialised_def using 1(2) xDef
ffold_init_def by auto
      moreover have fiE:"fullyInitialised e (Accounts sgas) (Stack sgas)"
        by (simp add: "*" "6.prems"(3))
      moreover have "fullyInitialised x (Accounts sgas) emptyStore"
      proof -
        have addX:"Address x = Address e" by (simp add: xDef)
        have typeX:"Type (Accounts sgas (Address x)) = Some (atype.Contract (Contract x))"
          using fiE addX cx unfolding fullyInitialised_def by force
        have epX:"ep $$ Contract x = Some (ct, fb)" using "1"(2) cx by simp
        have mapX:"∀ id v. (ct $$ id = Some (Var v)) = (Denvalue x $$ id = Some (type.Storage v,
Storeloc id))"
          using xDef unfolding ffold_init_def
          using ffoldInit_var_storage_mapping_eq[of ct "Address e" "Contract e" "Sender e" "Svalue e"
x]
          by blast
        have locX:"∀ id v loc. Denvalue x $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
        proof (intro allI impI)
          fix id v loc
          assume hx:"Denvalue x $$ id = Some (type.Storage v, Storeloc loc)"
          have "(Denvalue x $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v,
Storeloc loc) = Storeloc id)"
            using xDef unfolding ffold_init_def
            using ffoldInitAllLocsStorage[
              of ct "e" "e" "e" "Svalue e" "fmdom ct"
            ] by simp
          then show "id = loc" using hx by simp
        qed
        have ptrX:"∀ t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue x) ∧ accessStore l emptyStore = Some
(KStoptr p) ⟶
            (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue x) ∧ CompStoType t' t l' p)"
          by (simp add: accessStore_def emptyStore_def)
        show ?thesis
          unfolding fullyInitialised_def using typeX epX mapX locX ptrX by blast
      qed
      ultimately show ?thesis using exprTypeconInduct(4)[of False fp xe x emptyTypedStore emptyStore
"Memory sgas" e cd "sgas" "(state.Gas st - costs (INVOKE i xe) e cd st)"
          e_l cd_l k_l m_l g] a27 6(2) * sameADD by auto
    qed
    then have a40:" TypeSafe e_l (Accounts (st(|Gas := g, Stack := k_l, Memory := m_l|))) (Stack (st(|Gas :=
g, Stack := k_l, Memory := m_l|)))
     (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) (Storage (st(|Gas := g, Stack := k_l, Memory :=
m_l|))) cd_l" using * by auto

    have typeMstML:"∀ locs t. accessTypeStore locs (Memory st) = Some t ⟶ accessTypeStore locs (m_l) =
Some t "
    proof intros
      fix locs t
      assume in1:"accessTypeStore locs (Memory st) = Some t"
```

```
      then have "∃v. accessStore locs (Memory st) = Some v" using 6(2) unfolding TypeSafe_def
SomeValSomeTyp_def by auto
      then have "∃i. i < Toploc (Memory sgas) ∧ LSubPrefL2 locs (ShowL_nat i)"
        using 6(2) unfolding TypeSafe_def lessThanTopLocs_def using memSGas by presburger
      then have "accessTypeStore locs (Memory sgas) = accessTypeStore locs m_l"
        using link unfolding ncpOMemInDMem_def by blast
      then show "accessTypeStore locs m_l = Some t" using in1 memSGas by auto
    qed


    have a44:"(Address e_l) = (Address e)" using xDef a27
      using emptyEnv_address ffold_init_ad ffold_init_def msel_ssel_expr_load_rexp_gas(4) by presburger
    moreover have a45:"Contract e_l = Contract e" using xDef a27
      using emptyEnv_members ffold_init_contract ffold_init_def msel_ssel_expr_load_rexp_gas(4) by
presburger
    ultimately have cc1:"(∃c. Type (Accounts (st(|Gas := g, Stack := k_l, Memory := m_l|)) (Address
        e_l)) = Some (atype.Contract c) ∧ Contract e_l = c)"
      using 6(4) unfolding fullyInitialised_def by simp

    have a60:"∀id x'. Denvalue x $$ id = Some x' ⟶ Denvalue e_l $$ id = Some x'"
      using load_denval_existing_remain(4)[of False fp xe x emptyTypedStore emptyStore "(Memory sgas)"
e cd sgas "(state.Gas st - costs
          (INVOKE i xe) e cd st)" e_l cd_l k_l m_l g] a27 by blast


    have cc3:"(∀t l p.
        (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_l) ⟶
        accessStore l (Stack (st(|Gas := g, Stack := k_l, Memory := m_l|))) = Some (KStoptr p) ⟶
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_l) ∧ CompStoType t' t l' p))"
      using fullyInitialised_def link by auto

    have a50:" fullyInitialised e_l (Accounts (st(|Gas := g, Stack := k_l, Memory := m_l|))) (Stack
        (st(|Gas := g, Stack := k_l, Memory := m_l|)))" using 6(4)
        cc1 fullyInitialised_def link by auto
    have cc10:"applyf Stack (sgas(|Gas := g|)) = Normal ((Stack st), sgas(|Gas := g|))" using * by simp
    have IH1:"TypeSafe e_l (Accounts st'') (Stack st'') (Memory st'') (Storage st'') cd_l ∧
fullyInitialised e_l (Accounts st'') (Stack st'')
              ∧
    (∀locs t. accessTypeStore locs (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) = Some t ⟶
accessTypeStore locs (Memory st'') = Some t) ∧
(∀locs v. accessStore locs (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) = Some (MPointer v)
    ⟶ (∃v'. accessStore locs (Memory st'') = Some (MPointer v'))) ∧
    (∀locs v. accessStore locs (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) = Some (MValue v)
        ⟶ (∃v'. accessStore locs (Memory st'') = Some (MValue v'))) ∧
(∀i loc.
        i < Toploc (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
        accessStore loc (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) = None ⟶ accessStore loc
(Memory st'') = None) ∧
Toploc (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) ≤ Toploc (Memory st'')"
      using 6(1)[OF a5 a6 a10 _ a20 _ xDef a25 a30 _ _ _ cc10 _ a40 1(5) a50] * unfolding
StateInvariant_def by simp
    have cc20:"(Memory st') = (Memory st'') ∧ (Accounts st'') = (Accounts st') ∧ (Storage st'') =
(Storage st')" using 1(6) by simp
    have stackOver:"(Stack st') = (Stack st)" using 1(6) by auto


    show ?thesis unfolding TypeSafe_def StateInvariant_def
    proof intros
      show " AddressTypes (Accounts st')" using IH1 cc20 unfolding TypeSafe_def by auto
    next
      show "safeContract (Accounts st') (Storage st')" using IH1 cc20 unfolding TypeSafe_def by auto
    next
      show "unique_locations (Denvalue e)" using 6(2) unfolding TypeSafe_def by blast
    next
```

```
      show "compPointers (Stack st') (Denvalue e)" using 6(2) unfolding TypeSafe_def using cc20
        using stackOver by argo
    next
      show "balanceTypes (Accounts st')" using IH1 cc20 unfolding TypeSafe_def by auto
    next
      have "envAddressesWellFormed e" using 6(2) unfolding TypeSafe_def by blast
      then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
    next
      show "svalueTypes (Svalue e)" using 6(2) unfolding TypeSafe_def by blast
    next
      show "lessThanTopLocs (Stack st')" using 6(2) 1(6) unfolding TypeSafe_def by simp
    next
      show "lessThanTopLocs cd" using 6(2) unfolding TypeSafe_def by blast
    next
      show "lessThanTopLocs (Memory st')" using IH1 cc20 unfolding TypeSafe_def by auto
    next
      show " typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
        unfolding typeCompat_def
      proof intros
        fix tIn lIn
        assume in1:"(tIn, lIn) |∈| fmran (Denvalue e)"
        show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e)) cd tIn lIn"
        proof (split denvalue.split, intros)
          fix x1 assume stackLoc:"lIn = Stackloc x1"
          show "case accessStore x1 (Stack st') of None ⇒ False
              | Some (KValue val) ⇒ (case tIn of Value typ ⇒ typeCon typ val | _ ⇒ False)
              | Some (KCDptr stloc) ⇒ (case tIn of Calldata struct ⇒ MCon struct cd stloc | _ ⇒
False)
              | Some (KMemptr stloc) ⇒ (case tIn of type.Memory struct ⇒ MCon struct (Memory st')
stloc | _ ⇒ False)
              | Some (KStoptr stloc) ⇒ (case tIn of type.Storage struct ⇒ SCon struct stloc (Storage
st' (Address e)) | _ ⇒ False)"
          proof (split option.split, intros)
            assume "accessStore x1 (Stack st') = None"
            then show False using 6(2) unfolding TypeSafe_def typeCompat_def using stackOver in1
              by (metis "6.prems"(1) option.discI stackLoc typeSafeAllStacklocsExist)
          next
            fix x2 assume Some:"accessStore x1 (Stack st') = Some x2"
            then have SomeOld:"accessStore x1 (Stack st) = Some x2" using stackOver by simp
            show "case x2 of KValue val ⇒ (case tIn of Value typ ⇒ typeCon typ val | _ ⇒ False)
          | KCDptr stloc ⇒ (case tIn of Calldata struct ⇒ MCon struct cd stloc | _ ⇒ False)
          | KMemptr stloc ⇒ (case tIn of type.Memory struct ⇒ MCon struct (Memory st') stloc | _ ⇒
False)
          | KStoptr stloc ⇒ (case tIn of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address e)) | _ ⇒ False)"
            proof(split stackvalue.split, intros)
              fix x1 assume KValue:"x2 = KValue x1"
              then show "case tIn of Value typ ⇒ typeCon typ x1 | _ ⇒ False"
                using 6(2) unfolding TypeSafe_def typeCompat_def using stackLoc stackOver in1 Some
SomeOld by force
            next
              fix x2a assume KCDPtr:"x2 = KCDptr x2a"
              then show "case tIn of Calldata struct ⇒ MCon struct cd x2a | _ ⇒ False"
                using 6(2) unfolding TypeSafe_def typeCompat_def using stackLoc stackOver in1 Some
SomeOld by force
            next
              fix x3 assume KMemPtr:"x2 = KMemptr x3"
              then obtain struct where structDef:"tIn = type.Memory struct" using 6(2) unfolding
TypeSafe_def typeCompat_def using in1 stackOver Some stackLoc by (cases tIn; fastforce)
              then have McOld:"MCon struct (Memory st) x3" using 6(2) unfolding TypeSafe_def
typeCompat_def using in1 stackOver Some stackLoc KMemPtr by (cases tIn; fastforce)
              then have "∃x i. accessStore x3 (Memory st) = Some x ∨ accessStore (hash x3 (ShowL$_{nat}$
i)) (Memory st) = Some x" using MCon_imps_Some by blast
              moreover have "lessThanTopLocs (Memory st)"using 6(2) unfolding TypeSafe_def by auto
```

```
            ultimately obtain tloc where tlocDef: "(tloc<Toploc (Memory st) ∧ LSubPrefL2 x3
(ShowL_nat tloc))"
              by (meson lessThanTopLocs_def MemLSubPrefL2_specific_imps_general)

            have "(∀ locs tp.
     accessTypeStore locs (Memory st) = Some tp ⟶
     (case accessStore locs (Memory st) of None ⇒ False
      | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st) locs ∧ tp = MTValue val ∧ accessTypeStore locs
(Memory st) = Some tp
      | Some (MPointer p) ⇒ ∃ len arr. MCon tp (Memory st) p ∧ tp = MTArray len arr
∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st) = Some arr)))"
              using 6(2) unfolding TypeSafe_def subPrefixStructuralConsistency_def using McOld by
auto
            have IH1Link:"(∀ locs tp.
      accessTypeStore locs (Memory st') = Some tp ⟶
      (case accessStore locs (Memory st') of None ⇒ False
       | Some (MValue v) ⇒ ∃ val. MCon tp (Memory st') locs ∧ tp = MTValue val ∧ accessTypeStore locs
(Memory st') = Some tp
       | Some (MPointer p) ⇒ ∃ len arr. MCon tp (Memory st') p ∧ tp = MTArray len arr
∧ (∀ i<len. accessTypeStore (hash p (ShowL_nat i)) (Memory st') = Some arr)))"
              using IH1 cc20 unfolding TypeSafe_def subPrefixStructuralConsistency_def by presburger
            have mlFinalLink:"(∀ locs t. accessTypeStore locs (m_l) = Some t ⟶ accessTypeStore locs
(Memory st') = Some t)"
              using IH1 cc20 by simp
            have linkS:"(∀ i loc. i < Toploc (Memory sgas) ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
                    accessStore loc (Memory sgas) = accessStore loc m_l
                  ∧ accessTypeStore loc (Memory sgas) = accessTypeStore loc m_l)"
              using link unfolding ncpOMemInDMem_def by blast
            then have oldLoadedLink:"accessStore x3 (Memory st) = accessStore x3 m_l
                  ∧ accessTypeStore x3 (Memory st) = accessTypeStore x3 m_l" using tlocDef memSGas by
simp
            have SameVAccess:"(∀ locs v. accessStore locs (m_l) = Some (MPointer v) ⟶ (∃ v'.
accessStore locs (Memory st') = Some (MPointer v'))) ∧
  (∀ locs v. accessStore locs ((m_l)) = Some (MValue v) ⟶ (∃ v'. accessStore locs (Memory st') = Some
(MValue v')))" using IH1 cc20 by auto
            have "MCon struct (Memory st') x3 "
            proof(cases struct)
              case (MTArray x11 x12)
              then have accTypesOld:"∀ i<x11. accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) =
Some x12"
                using 6(2) unfolding TypeSafe_def denvalueTypeCorrectness_def
                using in1 Some stackOver KMemPtr structDef stackLoc by fastforce
              then have oldVals:"(∀ i<x11.
                                  case accessStore (hash x3 (ShowL_nat i)) (Memory st) of None ⇒ False
                                    | Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue
typ
                                        ⇒ MCon x12 (Memory st) (hash x3 (ShowL_nat i)))
                                    | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12
(Memory st) loc2 | MTValue val ⇒ False)) ∧
                                    (∃ p. accessStore x3 (Memory st) = Some (MPointer p) ∨ accessStore x3
(Memory st) = None)"
                using McOld MCon.simps(2)[of x11 x12 "Memory st" x3] MTArray by simp
              have "∀ i<x11. case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
                            | Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue typ ⇒
MCon
                                x12 (Memory st') (hash x3 (ShowL_nat i)))
                           | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12
(Memory st') loc2 | MTValue val ⇒ False)"
              proof intros
                fix i
                assume in1:"i<x11"

                then consider (ptr) p x11' x12' where "accessStore (hash x3 (ShowL_nat i)) (Memory
st) = Some (MPointer p)
```

```
                                                ∧ x12 = MTArray x11' x12'"
                  | (val) v vt where "accessStore (hash x3 (ShowL_nat i)) (Memory st) = Some (MValue
v)
                                          ∧ x12 = MTValue vt" using oldVals
                    by (metis MCon.simps(2) MTArray McOld mcon_accessStore)
                  then show "case accessStore (hash x3 (ShowL_nat i)) (Memory st') of None ⇒ False
                       | Some (MValue val) ⇒ (case x12 of MTArray l a ⇒ False | MTValue typ ⇒ MCon
x12 (Memory st') (hash x3 (ShowL_nat i)))
                       | Some (MPointer loc2) ⇒ (case x12 of MTArray len' arr' ⇒ MCon x12 (Memory
st') loc2 | MTValue val ⇒ False) "
                  proof(cases)
                    case ptr
                    then obtain t where accNew:"accessStore (hash x3 (ShowL_nat i)) (Memory st') = Some
(MPointer t)"
                      using linkS tlocDef memSGas SameVAccess
                      by (metis LSubPrefL2_def Not_Sub_More_Specific)
                    have "accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some (MTArray x11'
x12')"
                      using accTypesOld ptr in1 by simp
                    then have "accessTypeStore (hash x3 (ShowL_nat i)) (Memory st') = Some (MTArray x11'
x12')"
                      using mlFinalLink linkS tlocDef memSGas Not_Sub_More_Specific LSubPrefL2_def by
metis
                    then have "MCon (MTArray x11' x12') (Memory st') t" using IH1Link accNew by
fastforce
                    then show ?thesis using accNew ptr MTArray by simp
                  next
                    case val
                    then have accNew:"∃t. accessStore (hash x3 (ShowL_nat i)) (Memory st') = Some
(MValue t)"
                      using linkS tlocDef memSGas SameVAccess
                      by (metis LSubPrefL2_def Not_Sub_More_Specific)
                    have "accessTypeStore (hash x3 (ShowL_nat i)) (Memory st) = Some (MTValue vt)"
                      using accTypesOld val in1 by simp
                    then have "accessTypeStore (hash x3 (ShowL_nat i)) (Memory st') = Some (MTValue vt)"
                      using mlFinalLink linkS tlocDef memSGas Not_Sub_More_Specific LSubPrefL2_def by
metis
                    then have "MCon (MTValue vt) (Memory st') (hash x3 (ShowL_nat i))" using IH1Link
accNew by fastforce
                    then show ?thesis using MTArray accNew val by auto
                  qed

                qed
                moreover have "(∃p. accessStore x3 (Memory st') = Some (MPointer p) ∨ accessStore x3
(Memory st') = None)"
                proof -
                  have "∀i loc. i < Toploc m_l ∧ LSubPrefL2 loc (ShowL_nat i) ⟶
                      accessStore loc m_l = None ⟶ accessStore loc (Memory st') = None" using IH1 cc20
by auto
                  then show ?thesis using oldVals oldLoadedLink SameVAccess tlocDef
                    by (metis dual_order.strict_trans1 link memSGas)
                qed
                ultimately show ?thesis using MTArray McOld by auto
              next
                case (MTValue x2'')
                then have "(case accessStore x3 (Memory st) of None ⇒ False | Some (MValue t) ⇒
typeCon x2'' t | Some (MPointer t) ⇒ False)"
                  using McOld by simp
                then obtain t where accOld:"accessStore x3 (Memory st) = Some (MValue t)" by (auto
split:option.splits memoryvalue.splits)
                then have accNew:"∃t. accessStore x3 (Memory st') = Some (MValue t)" using
oldLoadedLink SameVAccess by auto
                have "accessTypeStore x3 (Memory st) = Some (MTValue x2'')"
                  using 6(2) unfolding TypeSafe_def denvalueTypeCorrectness_def using in1 Some
```

```
stackOver KMemPtr structDef MTValue
                using stackLoc by force
              then have "accessTypeStore x3 (Memory st') = Some (MTValue x2'')" using mlFinalLink
oldLoadedLink by auto
              then have "MCon (MTValue x2'') (Memory st') x3" using IH1Link accNew by force
              then show ?thesis using MTValue by simp
            qed


            then show "case tIn of type.Memory struct ⇒ MCon struct (Memory st') x3 | _ ⇒ False"
using structDef by simp
          next
            fix x4 assume KStoPtr:"x2 = KStoptr x4"

            then have accO:"accessStore x1 (Stack st) = Some( KStoptr x4)"
              using Some 1 by auto
            then obtain struct where structDef: "tIn = type.Storage struct"
              using 6(2) unfolding TypeSafe_def typeCompat_def using Some KStoPtr stackLoc in1 Some
SomeOld
              by (cases tIn) fastforce+
            obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x4)"
              using 6(4) using accO in1 unfolding fullyInitialised_def
              using stackLoc structDef by blast

            then have "SCon tprnt lprnt (Storage st'' (Address e))"
              using IH1 unfolding StateInvariant_def TypeSafe_def safeContract_def
              using cx a44 a45
              by (metis (mono_tags, lifting) "6.prems"(3) fi_denvalue_storeloc_to_contract_var
fullyInitialised_def)

            then have "SCon struct x4 (Storage st'' (Address e))"
              using tprntDef SCon_imps_sublocs by blast
            then show "case tIn of type.Storage struct ⇒ SCon struct x4 (Storage st' (Address e)) |
_ ⇒ False "
              using IH1 cc20 stackLoc stackOver in1 Some SomeOld structDef by simp
          qed
        qed
      next
        fix x2 assume storeLoc:"lIn = Storeloc x2"
        then obtain struct where structDef: "tIn = type.Storage struct"
          using 6(2) unfolding TypeSafe_def typeCompat_def using in1
          by (cases tIn) fastforce+
        obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x2)"
          using 6(4) using in1 unfolding fullyInitialised_def
          using structDef
          using CompStoType_sameLocNdTyp storeLoc by auto

        then have "SCon tprnt lprnt (Storage st'' (Address e))"
          using IH1 unfolding StateInvariant_def TypeSafe_def safeContract_def
          using cx a44 a45
          by (metis (mono_tags, lifting) "6.prems"(3) fi_denvalue_storeloc_to_contract_var
fullyInitialised_def)


        then have "SCon struct x2 (Storage st'' (Address e))"
          using tprntDef SCon_imps_sublocs by blast
        then show "case tIn of type.Storage struct ⇒ SCon struct x2 (Storage st' (Address e)) | _
⇒ False "
          using IH1 cc20 storeLoc stackOver in1 structDef by simp
      qed
    qed
  next
```

```
      show "fullyInitialised e (Accounts st') (Stack st') " using 6(4) 1(6) cc20 unfolding
fullyInitialised_def
        using IH1 <Address e_l = Address e> a45 fullyInitialised_def by auto
    next
      show "denvalueTypeCorrectness e (Stack st') (Memory st')" unfolding denvalueTypeCorrectness_def
      proof intros
        fix t l ptr_loc sub_loc
        assume in1:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') =
Some (KMemptr ptr_loc)"
        then have "accessStore l (Stack st) = Some (KMemptr ptr_loc)" using 1 by simp

        have "(∀ tp' locs p i.
       (type.Memory tp', Stackloc locs) |∈| fmran (Denvalue e_l) ∧ accessStore locs k_l = Some (KMemptr
p) ∧ i <
           Toploc (Memory sgas) ∧ LSubPrefL2 p (ShowL_nat i) ⟶
       (∃ tp'' loc2 p'.
           (type.Memory tp'', Stackloc loc2) |∈| fmran (Denvalue e) ∧
           accessStore loc2 (Stack sgas) = Some (KMemptr p') ∧
           (p' = p ∧ tp'' = tp' ∨ (∃ len arr. p' ≠ p ∧ tp'' = MTArray len arr ∧ CompMemType (Memory
sgas) len arr tp' p' p))))"
          using link unfolding ncpDenvalueLimit_def by blast
        have "(∀ t l ptr_loc sub_loc.
       (type.Memory t, Stackloc l) |∈| fmran (Denvalue e_l) ∧ accessStore l (Stack st'') = Some (KMemptr
ptr_loc) ⟶
       (case t of
        MTArray len arr ⇒
          (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st'') = Some arr)
        | MTValue val ⇒ accessTypeStore ptr_loc (Memory st'') = Some (MTValue val)))"
          using IH1 unfolding TypeSafe_def denvalueTypeCorrectness_def by blast
        have "(case t of MTArray len arr ⇒ ∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory
st) = Some arr
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st) = Some (MTValue val))"
          using 6(2) in1 1 unfolding TypeSafe_def denvalueTypeCorrectness_def by simp
        then show "case t of
       MTArray len arr ⇒
         (∀ i<len. accessTypeStore (hash ptr_loc (ShowL_nat i)) (Memory st') = Some arr)
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
          using IH1 memSGas typeMstML cc20 by (cases t; simp)
      qed
    next
      show "SomeValSomeTyp (Memory st')"
        using IH1 TypeSafe_def cc20 by auto
    next
      show "subPrefixStructuralConsistency (Memory st')" using IH1 TypeSafe_def cc20 by simp
    next
      show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t "
        using IH1 cc20 typeMstML memSGas by auto
    next
      fix locs v
      assume in1:"accessStore locs (Memory st) = Some (MPointer v)"
      then have "∃ i. i < Toploc (Memory sgas) ∧ LSubPrefL2 locs (ShowL_nat i)"
        using memSGas 6(2) unfolding TypeSafe_def lessThanTopLocs_def by auto
      then have "accessStore locs (m_l) = Some (MPointer v)"
        using in1 link memSGas unfolding ncpOMemInDMem_def by auto
      then show " ∃ v'. accessStore locs (Memory st') = Some (MPointer v')"
        using IH1 cc20 by simp
    next
      fix locs v
      assume in1:"accessStore locs (Memory st) = Some (MValue v)"
      then have "∃ i. i < Toploc (Memory sgas) ∧ LSubPrefL2 locs (ShowL_nat i)"
        using memSGas 6(2) unfolding TypeSafe_def lessThanTopLocs_def by auto
      then have "accessStore locs (m_l) = Some (MValue v)"
        using in1 link memSGas unfolding ncpOMemInDMem_def by auto
```

```
      then show " ∃v'. accessStore locs (Memory st') = Some (MValue v')"
        using IH1 cc20 by simp
    next
      fix i loc
      assume in1:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) "
        and in2:"accessStore loc (Memory st) = None"
      then have "accessStore loc (m_l) = None"
        using in1 link memSGas unfolding ncpOMemInDMem_def by auto
      moreover have "i < Toploc (Memory (st(|Gas := g, Stack := k_l, Memory := m_l|))) ∧ LSubPrefL2 loc
(ShowL_nat i)"
        using in1 link memSGas by simp
      ultimately show "accessStore loc (Memory st') = None"
        using IH1 cc20 by auto
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')" using memSGas link IH1 cc20 by simp
    qed
  qed
next
  case (7 ad i xe val e cd st)
  show ?case
  proof(cases rule:external[OF 7(4)])
    case (1 adv c g ct cn fb' v t g' v' fp f e_l cd_l k_l m_l g'' acc st'')
    then obtain cc :: environment where ccDef: "c = Contract cc"
      by (metis Environment.select_convs(2))

    then have a5:"assert Gas (λst. costs (EXTERNAL ad i xe val) e cd st < state.Gas st) st = Normal
((), st)" using 1 by simp
    then obtain sgas where *:"sgas = st(|Gas := state.Gas st - costs (EXTERNAL ad i xe val) e cd st|)"
by simp
    then have a6:"modify (λst. st(|Gas := state.Gas st - costs (EXTERNAL ad i xe val) e cd st|)) st =
Normal ((), sgas)" by simp
    obtain expGas where a10:" toState (local.expr ad e cd) sgas = Normal ((KValue adv, Value TAddr),
expGas)" using 1(2) a6 * by simp
    then have a20:"(case (KValue adv, Value TAddr) of (KValue adv, Value TAddr) ⇒ return adv | (KValue
adv, Value _) ⇒ throw Err
                  | (KValue adv, _) ⇒ throw Err | (_, b) ⇒ throw Err) expGas =
    Normal (adv, expGas)" by simp
    then obtain vd where a30:"assert Err (λ_. adv ≠ Address e) expGas = Normal (vd, expGas)" using 1
by simp
    then have a40:"(case Type (Accounts expGas adv) of None ⇒ throw Err expGas
              | Some EOA ⇒ throw Err expGas
              | Some (atype.Contract c) ⇒ return c expGas) = Normal (c, expGas)" using 1 a10 * by
auto
    then have a50:"option Err (λ_. ep $$ c) expGas = Normal ((ct, cn, fb'), expGas)" using 1 by simp
    then have a60:"toState (local.expr val e cd) expGas = Normal ((KValue v, Value t), expGas(|Gas :=
g'|))" using 1 a10 * by auto
    then have a70:"(case (KValue v, Value t) of (KValue v, Value t) ⇒ return (v, t)
                  | (KValue v, _) ⇒ throw Err
                  | (_, b) ⇒ throw Err) (expGas(|Gas := g'|)) = Normal ((v,t), expGas(|Gas := g'|))"
using 1 a10 * by simp
    then have a80:"option Err (λ_. convert t (TUInt b256) v) (expGas(|Gas := g'|)) = Normal (v',
expGas(|Gas := g'|))" using 1 by simp
    then obtain loaded where a90:"loaded = ffold_init ct (emptyEnv adv c (Address e) v') (fmdom ct)"
using 1(9) by blast
    then have a100:"toState (local.load True fp xe loaded emptyTypedStore emptyStore emptyTypedStore e
cd) (expGas(|Gas := g'|))
        = Normal ((e_l, cd_l, k_l, m_l), expGas(|Gas := g''|))"
      using 1 * a10 by auto
    then have a110:"option Err (λst. transfer (Address e) adv v' (Accounts st)) (expGas(|Gas := g''|)) =
Normal (acc, expGas(|Gas := g''|))"
      using 1 * a10 by auto
    then have a120:"modify (λst. st(|Accounts := acc, Stack := k_l, Memory := m_l|)) (expGas(|Gas :=
        g''|)) = Normal ((), (expGas(|Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l|)))"
      using 1 * a10 by auto
```

```
    have contractLoaded:"Contract e_l = c" using a90 a100
      using "1"(9) emptyEnv_members ffold_init_contract msel_ssel_expr_load_rexp_gas(4) by presburger
    have addressLoaded: "Address e_l = Address loaded" using a90 a100
      using "1"(9) ffold_init_def msel_ssel_expr_load_rexp_gas(4) by presburger

    have tsInitAcc:"∀ x y. TypeSafe loaded acc emptyStore emptyTypedStore (Storage st) emptyTypedStore
∧
       (Denvalue loaded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue loaded $$ x = Some y ⟶
          (∃ t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    proof -
      have sc:"safeContract acc (Storage st)"
        using 7(3) 1(10) transfer_type_same unfolding TypeSafe_def safeContract_def by fastforce
      have svv':"svalueTypes v'" unfolding svalueTypes_def
        using 1(6,7) 7(3) 7(5)
        using exprTypeconInduct(3)[of val e cd "(st(|Gas := g|))" g "KValue v" "Value t" g']
          typeSafeConvert[of t v "(TUInt b256)"] convertSame by auto
      have bt:"balanceTypes acc" using 1(10) unfolding balanceTypes_def
      proof -
        have old:"balanceTypes (Accounts (st(|Gas := g''|)))" using 7(3) unfolding TypeSafe_def by simp
        have accExp:"(case subBalance (Address e) v' (Accounts (st(|Gas := g''|))) of None ⇒ None | Some
x ⇒ addBalance adv v' x) = Some acc"
          using 1(10) unfolding transfer_def by blast
        have v'TC: "typeCon (TUInt b256) v'"
          using svv' unfolding svalueTypes_def by blast

        show "∀ adv'. typeCon (TUInt b256) (Bal (acc adv')) "
        proof
          fix adv'
          show "typeCon (TUInt b256) (Bal (acc adv')) "
          proof(cases "adv' = (Address e)")
            case True
            then have ns:"adv' ≠ adv" using 1 by blast
            then have tco:"typeCon (TUInt b256) (Bal ((Accounts (st(|Gas := g''|))) adv'))"
              using True old accExp balanceTypes_def by blast
            have "Bal (acc (Address e)) = ShowL_int (ReadL_int (Bal (Accounts (st(|Gas :=
                g''|)) (Address e))) - (ReadL_int v'))"
              using transfer_subRead[OF 1(10) _ v'TC ] True ns tco by auto
            then show ?thesis using True
              by (metis transfer_Bal_typeCon[unfolded transfer_def, OF accExp] old[unfolded
balanceTypes_def])
          next
            case False
            then show ?thesis
            proof(cases "adv' = adv")
              case True
              then have "typeCon (TUInt b256) (Bal (Accounts (st(|Gas := g''|)) adv))"
                using True old accExp balanceTypes_def by simp
              then have "Bal (acc adv) = (ShowL_int ((ReadL_int (Bal (Accounts (st(|Gas := g''|)) adv))) +
(ReadL_int v')))"
                using transfer_addRead[OF 1(10)_ v'TC ] 1(3) by blast
              then show ?thesis using True
                using "1"(10,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st(|Gas := g''|))
adv))› checkUInt_def transfer_val1 transfer_val2 by auto
            next
              case f2:False
              then show ?thesis using False transfer_eq[OF 1(10)] old
                using balanceTypes_def by force
            qed
          qed
        qed
      qed

      have ltp:"lessThanTopLocs emptyTypedStore" using typedEmptyTopLocs by simp
```

```
      have tp:"ep $$ Contract cc = Some (ct, cn, fb')" using ccDef 1(5) by simp
      have af1:"addressFormat adv" using 7(3) unfolding TypeSafe_def AddressTypes_def using 1 by (simp
split:option.splits atype.splits)
      have af2:"addressFormat (Address e)" using 7(3) unfolding TypeSafe_def AddressTypes_def using 1
by simp
      have af3:"AddressTypes acc" using 1(10) transfer_type_same 7(3) unfolding TypeSafe_def unfolding
AddressTypes_def by simp
      have tcAdvAcc:"Type (acc adv) = Some (atype.Contract (Contract cc))"
        using 1(10) a10 * a40 ccDef transfer_type_same by (auto split:option.splits atype.splits
result.splits)
      have svt:"SomeValSomeTyp emptyTypedStore" unfolding SomeValSomeTyp_def accessStore_def
accessTypeStore_def emptyTypedStore_def by auto
      have sp:"subPrefixStructuralConsistency emptyTypedStore" unfolding subPrefixStructuralConsistency_def
accessTypeStore_def emptyTypedStore_def by simp
      then have " ffold (init ct) (emptyEnv adv (Contract cc) (Address e) v') (fmdom ct) = loaded"
using a90 unfolding ffold_init_def using ccDef by simp
      then show ?thesis
        using ffoldInitTypeSafe[OF sc bt svv' ltp tp af1 af2 af3 sp svt tcAdvAcc,of "(fmdom ct)"] by
simp
    qed

    have tsInit:"∀x y. TypeSafe loaded (Accounts st) emptyStore emptyTypedStore (Storage st)
emptyTypedStore ∧
       (Denvalue loaded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue loaded $$ x = Some y ⟶
          (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    proof -
      have "safeContract (Accounts st) (Storage st)" using 7(3) unfolding TypeSafe_def AddressTypes_def
using 1 by simp
      moreover have svv':"svalueTypes v'" unfolding svalueTypes_def
        using 1(6,7) 7(3) 7(5)
        using exprTypeconInduct(3)[of val e cd "(st⦇Gas := g⦈)" g "KValue v" "Value t" g']
          typeSafeConvert[of t v "(TUInt b256)"] convertSame by auto
      moreover have "balanceTypes (Accounts st)" using 7(3) unfolding TypeSafe_def by simp
      moreover have "lessThanTopLocs emptyStore" using emptyTopLocs by simp
      moreover have "ep $$ Contract cc = Some (ct, cn, fb')" using ccDef 1(5) by simp
      moreover have "addressFormat adv" using 7(3) unfolding TypeSafe_def AddressTypes_def using 1 by
(simp split:option.splits atype.splits)
      moreover have "addressFormat (Sender e)" using 7(3) unfolding TypeSafe_def AddressTypes_def
using 1 by simp
      moreover have "AddressTypes (Accounts st)" using 7(3) unfolding TypeSafe_def by simp
      moreover have "SomeValSomeTyp emptyTypedStore" unfolding SomeValSomeTyp_def accessTypeStore_def
accessStore_def emptyTypedStore_def by simp
      moreover have sp:"subPrefixStructuralConsistency emptyTypedStore" unfolding
subPrefixStructuralConsistency_def accessTypeStore_def emptyTypedStore_def by simp
      moreover have "Type (acc adv) = Some (atype.Contract (Contract cc))"
        using 1(10) a10 * a40 ccDef transfer_type_same by (auto split:option.splits atype.splits
result.splits)

      ultimately show ?thesis
        using a90 ccDef unfolding ffold_init_def
        using ffoldInitTypeSafe[of acc "Storage st" v' emptyTypedStore cc ct "(cn,fb')" adv _ " (fmdom
ct)"]
        using TypeSafe_def tsInitAcc by presburger
    qed

    have contractL:"Contract loaded = c" using a90 by simp
    have addressL:"Address loaded = adv" using a90 by simp
    have fi1:"(∃c. Type (Accounts (st⦇Gas := g'⦈)) (Address loaded)) = Some (atype.Contract c) ∧
Contract loaded = c)" using contractL addressL 1 a10 * a40 by simp
    have "∃dud. ep $$ Contract loaded = Some (ct, dud)" using a50 contractL 1 a10 * by blast
    moreover have "∀id v. (ct $$ id = Some (Var v)) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id))"
      using a90 unfolding ffold_init_def using ffoldInit_var_storage_mapping_eq[of ct adv c "Address e"
v' loaded] by blast
```

```
    have "Type (Accounts (expGas⦇Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l⦈) (Address e_l))
=
        Type (Accounts (st⦇Gas := g'⦈) (Address loaded))"
      using addressLoaded 1(10) a10 * transfer_type_same by simp
    then have fi3:"(∃ c. Type (Accounts (expGas⦇Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l⦈)
        (Address e_l)) = Some (atype.Contract c) ∧ Contract e_l = c)"
      using contractLoaded fi1
      using contractL by presburger


    then have tsLoaded:"TypeSafe e_l (Accounts (st⦇Gas := g'⦈)) k_l m_l (Storage (st⦇Gas := g'⦈)) cd_l ∧
    fullyInitialised e_l (Accounts (st⦇Gas := g'⦈)) k_l"
    proof -
      have ld:"load True fp xe loaded emptyTypedStore emptyStore emptyTypedStore e cd (st⦇Gas := g'⦈)
g' = Normal ((e_l, cd_l, k_l, m_l), g'')" using 1
        using a90 ffold_init_def by presburger

      have "fullyInitialised loaded (Accounts (st⦇Gas := g'⦈)) emptyStore"
      proof -
        obtain c_fi where cfi:
          "Type (Accounts (st⦇Gas := g'⦈) (Address loaded)) = Some (atype.Contract c_fi) ∧ Contract
loaded = c_fi"
          using fi1 by blast
        obtain dud_fi where epfi:"ep $$ Contract loaded = Some (ct, dud_fi)"
          using a50 contractL 1 a10 * by blast
        have mapfi:"∀ id v. (ct $$ id = Some (Var v)) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id))"
          using a90 unfolding ffold_init_def using ffoldInit_var_storage_mapping_eq[of ct adv c
"Address e" v' loaded]
          by blast
        have locfi:"∀ id v loc. Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc"
        proof (intro allI impI)
          fix id v loc
          assume h:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc)"
          have hloc:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage
v, Storeloc loc) = Storeloc id"
            using tsInit by blast
          then show "id = loc" using h by simp
        qed
        have ptrfi:"∀ t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue loaded) ∧ accessStore l emptyStore = Some
(KStoptr p) ⟶
            (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue loaded) ∧ CompStoType t' t l'
p)"
          by (simp add: accessStore_def emptyStore_def)
        show ?thesis using cfi epfi mapfi locfi ptrfi
          using fullyInitialised_def by blast
      qed
      then show ?thesis
        using * 1(9) 7(3) tsInit 7(5) using fi1 ld
        using exprTypeconInduct(4)[of True fp xe loaded emptyTypedStore emptyStore emptyTypedStore e cd
"(st⦇Gas := g'⦈)" g' e_l cd_l k_l m_l g'']
        by simp
    qed


    then have ts1:"TypeSafe e_l (Accounts (expGas⦇Gas :=
        g'', Accounts := acc, Stack := k_l, Memory := m_l⦈)) (Stack (expGas⦇Gas := g'', Accounts := acc,
Stack :=
        k_l, Memory := m_l⦈))
     (Memory (expGas⦇Gas :=
```

```
        g'', Accounts := acc, Stack := k_l, Memory := m_l|))) (Storage (expGas(|Gas := g'', Accounts :=
acc, Stack
        := k_l, Memory := m_l|))) cd_l"
      using tsInitAcc a10 * 1 unfolding TypeSafe_def by auto

    have fiFinal:"fullyInitialised e_l (Accounts(expGas(|Gas :=
       g'', Accounts := acc, Stack := k_l, Memory := m_l|))) (Stack (expGas(|Gas := g'', Accounts := acc,
Stack :=
       k_l, Memory := m_l|)))"
    proof -
      from tsLoaded obtain c_fi ct_fi dud_fi where fiOld:"
          Type (Accounts (st(|Gas := g'|)) (Address e_l)) = Some (atype.Contract c_fi) ∧
          Contract e_l = c_fi ∧
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_l $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀id v loc. Denvalue e_l $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
          (∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_l) ∧ accessStore l k_l = Some (KStoptr
p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_l) ∧ CompStoType t' t l'
p))"
        unfolding fullyInitialised_def by blast
      have typeNew:"Type (Accounts (expGas(|Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l|))
(Address e_l))
            = Some (atype.Contract c_fi)"
        using fiOld addressLoaded
          ‹Type (Accounts (expGas(|Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l|)) (Address
              e_l)) = Type (Accounts (st(|Gas := g'|)) (Address loaded))›
        by simp
      have ptrNew:"∀t l p.
        (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_l) ∧
        accessStore l (Stack (expGas(|Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l|))) = Some
(KStoptr p) ⟶
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_l) ∧ CompStoType t' t l' p)"
        using fiOld by simp
      show ?thesis
        unfolding fullyInitialised_def using fiOld typeNew ptrNew by blast
    qed

    then have IH1:"StateInvariant e_l (expGas(|Gas := g'', Accounts := acc, Stack := k_l, Memory := m_l|))
st'' cd_l"
      using 7(1)[OF a5 a6 a10 a20 a30 a40 a50 _ _ a60 a70 _ a80 a90 1(8) _ _ _ _ a100 _ _ _ a110 _ _
a120 ts1
          ] 1(11) 1(2) a10 * by fastforce

    have accFinal:"Accounts st' = Accounts st''" using 1 by simp
    have storageFinal:"Storage st' = Storage st''" using 1 by simp

    have SafeC:"safeContract (Accounts st'') (Storage st'')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast
    then have "Storage st'' = Storage st'" using 1 by auto
    then have scNew:"∀e ct dud i tp. Type (Accounts
        st'' (Address e)) = Some (atype.Contract (environment.Contract e)) ∧ ep $$ Contract
(e::environment) = Some (ct, dud) ∧
            ct $$ i = Some (Var tp) ⟶ SCon tp i (Storage st'' (Address e))"
      using SafeC unfolding safeContract_def by simp
    have acc:"(∀adv. case Type (Accounts st adv) of None ⇒ True | Some EOA ⇒ True | Some
(atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep)"
      using 7(3) unfolding TypeSafe_def AddressTypes_def by blast
    obtain cOld where "Type (Accounts st (Address e)) = Some (atype.Contract cOld) ∧ Contract e =
cOld"
      using 7(5) unfolding fullyInitialised_def by simp
    then obtain ctO dudO where ctoDef: "ep $$ Contract e = Some (ctO, dudO)" using acc
```

```
      using "7.prems"(3) typesafe_base.fullyInitialised_def typesafe_base_axioms by blast
    have fi2:"∀ id v. (ctO $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))"
      using 7(5) ctoDef unfolding fullyInitialised_def by auto
    have fiLoc:"∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
      using 7(5) unfolding fullyInitialised_def by blast

    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def using
1(12) by simp
      next
        show "safeContract (Accounts st') (Storage st')" using IH1 storageFinal 1 unfolding
StateInvariant_def TypeSafe_def by auto
      next
        show "unique_locations (Denvalue e)" using "7.prems"(1) typeSafeUnique by blast
      next
        have old:" compPointers (Stack st'') (Denvalue e_l)"
          using IH1 unfolding StateInvariant_def TypeSafe_def by blast
        show "compPointers (Stack st') (Denvalue e)" unfolding compPointers_def
        proof intros
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume a0:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then have a5:"accessStore l1' (Stack st') = accessStore l1' (Stack st)" using 1 by simp
          have a10:"accessStore l2' (Stack st') = accessStore l2' (Stack st)" using 1 by auto
          show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
                else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
            using 7(3) unfolding TypeSafe_def compPointers_def using a0 a5 a10 by simp
        qed
      next
        show "balanceTypes (Accounts st')" using accFinal IH1 unfolding StateInvariant_def
TypeSafe_def by auto
      next
        have "envAddressesWellFormed e" using "7.prems"(1) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using 1(12) "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs cd" using "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using 1(12) "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume tc1:"(t, l) |∈| fmran (Denvalue e)"
          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e)) cd t l"
          proof(cases l)
            case (Stackloc x1)
            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using tc1 Stackloc using 7(3) 1(12) unfolding TypeSafe_def
typeCompat_def by force
            next
```

```
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 1(12) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KCDptr x2)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 1(12) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KMemptr x3)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 1(12) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KStoptr x4)
                then have acc0:"accessStore x1 (Stack st) = Some( KStoptr x4)"
                  using Some 1 by auto
                then obtain struct where structDef: "t = type.Storage struct"
                  using 7(3) unfolding TypeSafe_def typeCompat_def using tc1 Some KStoptr Stackloc
                  by (cases t) fastforce+
                obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x4)"
                  using 7(5) using acc0 tc1 unfolding fullyInitialised_def
                  using Stackloc structDef by blast

                then have "SCon tprnt lprnt (Storage st'' (Address e))"
                proof -
                  from tprntDef obtain kpr where kprDef:"Denvalue e $$ kpr = Some (type.Storage
tprnt, Storeloc lprnt)"
                    using fmlookup_ran_iff by blast
                  have "kpr = lprnt" using fiLoc kprDef by blast
                  hence denL:"Denvalue e $$ lprnt = Some (type.Storage tprnt, Storeloc lprnt)" using
kprDef by simp
                  have "ct0 $$ lprnt = Some (Var tprnt)" using fi2 denL by blast
                  moreover have typeEFinal:"Type (Accounts st'' (Address e)) = Some (atype.Contract
(Contract e))"
                    using 7(5) atype_same "7.prems"(2,3) unfolding fullyInitialised_def
                    by (metis accFinal)
                  ultimately show ?thesis using scNew ctoDef by blast
                qed
                then have "SCon struct x4 (Storage st'' (Address e))"
                  using tprntDef SCon_imps_sublocs by blast
                then show ?thesis using Stackloc Some KStoptr structDef storageFinal by simp
              qed
            qed
          next
            case (Storeloc x2)
            then obtain struct where structDef: "t = type.Storage struct"
              using 7(3) tc1 unfolding TypeSafe_def typeCompat_def by (cases t,fastforce+)

            then have "SCon struct x2 (Storage st'' (Address e))"
            proof -
              have inRan:"(type.Storage struct, Storeloc x2) |∈| fmran (Denvalue e)"
                using tc1 Storeloc structDef by simp
              then obtain kpr where kprDef:"Denvalue e $$ kpr = Some (type.Storage struct, Storeloc
x2)"
                using fmlookup_ran_iff by metis
              have "kpr = x2" using fiLoc kprDef by blast
              hence denX2:"Denvalue e $$ x2 = Some (type.Storage struct, Storeloc x2)" using kprDef by
simp
              have "ct0 $$ x2 = Some (Var struct)" using fi2 denX2 by blast
              moreover have typeEFinal:"Type (Accounts st'' (Address e)) = Some (atype.Contract
(Contract e))"
```

```
            using 7(5) atype_same "7.prems"(2,3) unfolding fullyInitialised_def
            by (metis accFinal)
          ultimately show ?thesis using scNew ctoDef by blast
        qed
        then show ?thesis using Storeloc structDef
          by (simp add: storageFinal)
      qed
    qed
  next
    show "denvalueTypeCorrectness e (Stack st') (Memory st')" using 7(3) 1(12) unfolding
TypeSafe_def by simp
  next
    show "subPrefixStructuralConsistency (Memory st')" using 7(3) 1(12) unfolding TypeSafe_def by
simp
  next
    show "SomeValSomeTyp (Memory st')" using 7(3) 1(12) unfolding TypeSafe_def by simp
  qed
next
  have fiOld:"
    ∃c ct dud.
      Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
      Contract e = c ∧
      ep $$ c = Some (ct, dud) ∧
      (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc id)))
∧
      (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
      (∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))"
    using 7(5) unfolding fullyInitialised_def by blast
  then obtain c_fi ct_fi dud_fi where fiC:"
      Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
      Contract e = c_fi ∧
      ep $$ c_fi = Some (ct_fi, dud_fi) ∧
      (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
      (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
    and fiPtr:"∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
    by blast
  have typeNew:"Type (Accounts st' (Address e)) = Some (atype.Contract c_fi)"
    using fiC atype_same "7.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
  have ptrNew:"∀t l p.
       (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
       (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
    using 1 fiFinal 7(5) unfolding fullyInitialised_def by auto
  show "fullyInitialised e (Accounts st') (Stack st')"
    unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
next
  show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
    using 7(3) 1(12) by simp
next
  show "⋀locs v.
   accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory st') = Some
(MPointer v')"
    using 1(12) by simp
next
  show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
```

```
st') = Some (MValue v')"
        using 1(12) by simp
    next
      show "⋀i loc.
       i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹
       accessStore loc (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using 1(12) by simp
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')" using 1(12) by simp
    qed
  next
    case (2 adv c g ct cn fb' v t g' v' acc st'')
    then obtain cc :: environment where ccDef: "c = Contract cc"
      by (metis Environment.select_convs(2))

    then have a5:"assert Gas (λst. costs (EXTERNAL ad i xe val) e cd st < state.Gas st) st = Normal
((), st)" using 2 by simp
    then obtain sgas where *:"sgas = st⦇Gas := state.Gas st - costs (EXTERNAL ad i xe val) e cd st⦈"
by simp
    then have a6:"modify (λst. st⦇Gas := state.Gas st - costs (EXTERNAL ad i xe val) e cd st⦈) st =
Normal ((), sgas)" by simp
    obtain expGas where a10:" toState (local.expr ad e cd) sgas = Normal ((KValue adv, Value TAddr),
expGas)" using 2(2) a6 * by simp
    then have a20:"(case (KValue adv, Value TAddr) of (KValue adv, Value TAddr) ⇒ return adv | (KValue
adv, Value _) ⇒ throw Err
                  | (KValue adv, _) ⇒ throw Err | (_, b) ⇒ throw Err) expGas =
    Normal (adv, expGas)" by simp
    then obtain vd where a30:"assert Err (λ_. adv ≠ Address e) expGas = Normal (vd, expGas)" using 2
by simp
    then have a40:"(case Type (Accounts expGas adv) of None ⇒ throw Err expGas
               | Some EOA ⇒ throw Err expGas
               | Some (atype.Contract c) ⇒ return c expGas) = Normal (c, expGas)" using 2 a10 * by
auto
    then have a50:"option Err (λ_. ep $$ c) expGas = Normal ((ct, cn, fb'), expGas)" using 2 by simp
    then have a60:"toState (local.expr val e cd) expGas = Normal ((KValue v, Value t), expGas⦇Gas :=
g'⦈)" using 2 a10 * by auto
    then have a70:"(case (KValue v, Value t) of (KValue v, Value t) ⇒ return (v, t)
                  | (KValue v, _) ⇒ throw Err
                  | (_, b) ⇒ throw Err) (expGas⦇Gas := g'⦈) = Normal ((v,t), expGas⦇Gas := g'⦈)"
using 2 a10 * by simp
    then have a80:"option Err (λ_. convert t (TUInt b256) v) (expGas⦇Gas := g'⦈) = Normal (v',
expGas⦇Gas := g'⦈)" using 2 by simp
    then obtain loaded where a90:"loaded = ffold_init ct (emptyEnv adv c (Address e) v') (fmdom ct)"
using 2(9) by blast

    then have a110:"option Err (λst. transfer (Address e) adv v' (Accounts st)) (expGas⦇Gas := g'⦈) =
Normal (acc, expGas⦇Gas := g'⦈)"
      using 2 * a10 by auto


    have tsInitAcc:"∀x y. TypeSafe loaded acc emptyStore emptyTypedStore (Storage st) emptyTypedStore
∧
       (Denvalue loaded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue loaded $$ x = Some y ⟶
           (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    proof -
      have "safeContract (Accounts st) (Storage st)" using 7(3) unfolding TypeSafe_def AddressTypes_def
using 2 by simp
      moreover have "safeContract acc (state.Storage st)"
        using "2"(9) calculation safeContract_def transfer_type_same by auto
      moreover have svv':"svalueTypes v'" unfolding svalueTypes_def
        using 2(6,7) 7(3) 7(5)
        using exprTypeconInduct(3)[of val e cd "(st⦇Gas := g⦈)" g "KValue v" "Value t" g']
          typeSafeConvert[of t v "(TUInt b256)"] convertSame by auto
      moreover have "balanceTypes acc" using 2(10) unfolding balanceTypes_def
```

```
      proof -
        have old:"balanceTypes (Accounts (st⦇Gas := g'⦈))" using 7(3) unfolding TypeSafe_def by simp
        have accExp:"(case subBalance (Address e) v' (Accounts (st)) of None ⇒ None | Some x ⇒
addBalance adv v' x) = Some acc"
          using 2(9) unfolding transfer_def by blast
        have v'TC: "typeCon (TUInt b256) v'"
          using svv' unfolding svalueTypes_def by blast

        show "∀adv'. typeCon (TUInt b256) (Bal (acc adv')) "
        proof
          fix adv'
          show "typeCon (TUInt b256) (Bal (acc adv')) "
          proof(cases "adv' = (Address e)")
            case True
            then have ns:"adv' ≠ adv" using 2 by blast
            then have tco:"typeCon (TUInt b256) (Bal ((Accounts (st⦇Gas := g'⦈)) adv'))"
              using True old accExp balanceTypes_def by blast
            have "Bal (acc (Address e)) = ShowL_int (ReadL_int (Bal (Accounts (st⦇Gas := g'⦈)) (Address
e))) - (ReadL_int v'))"
              using transfer_subRead[OF 2(9) _ v'TC ] True ns tco by auto
            then show ?thesis using True

              using "2"(9) Read_ShowL_id checkUInt_def tco transfer_val3 v'TC by fastforce
          next
            case False
            then show ?thesis
            proof(cases "adv' = adv")
              case True
              then have "typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈)) adv))"
                using True old accExp balanceTypes_def by simp
              then have "Bal (acc adv) = (ShowL_int ((ReadL_int (Bal (Accounts (st⦇Gas := g'⦈)) adv))) +
(ReadL_int v'))"
                using transfer_addRead[OF 2(9)_ v'TC ] 2(3) by auto
              then show ?thesis using True
                using "2"(9,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈))
adv))› checkUInt_def transfer_val1 transfer_val2 by auto
            next
              case f2:False
              then show ?thesis using False transfer_eq[OF 2(9)] old
                using balanceTypes_def by force
            qed
          qed
        qed
      qed
      moreover have "lessThanTopLocs emptyStore" using emptyTopLocs by simp
      moreover have "ep $$ Contract cc = Some (ct, cn, fb')" using ccDef 2(5) by simp
      moreover have "addressFormat adv" using 7(3) unfolding TypeSafe_def AddressTypes_def using 2 by
(simp split:option.splits atype.splits)
      moreover have "addressFormat (Sender e)" using 7(3) unfolding TypeSafe_def AddressTypes_def
using 2 by simp
      moreover have "AddressTypes acc" using 2(9) transfer_type_same 7(3) unfolding TypeSafe_def un-
folding AddressTypes_def by simp
      moreover have "addressFormat (Address e)" using "7.prems"(1) unfolding TypeSafe_def by simp
      moreover have "SomeValSomeTyp emptyTypedStore" unfolding SomeValSomeTyp_def accessTypeStore_def
accessStore_def
        by (simp add: emptyTypedStore_def)
      moreover have sp:"subPrefixStructuralConsistency emptyTypedStore" unfolding
subPrefixStructuralConsistency_def accessTypeStore_def emptyTypedStore_def by simp
      moreover have "Type (acc adv) = Some (atype.Contract (Contract cc))"
        using 2(9) a10 * a40 ccDef transfer_type_same by (auto split:option.splits atype.splits
result.splits)

      ultimately show ?thesis
        using a90 ccDef unfolding ffold_init_def using ffoldInitTypeSafe[of acc "Storage st" v'
```

```
emptyTypedStore cc ct "(cn,fb')" adv "(Address e)" " (fmdom ct)"]
        using typedEmptyTopLocs by blast
    qed

    have contractL:"Contract loaded = c" using a90 by simp
    have addressL:"Address loaded = adv" using a90 by simp
    have fi1:"(∃c. Type (Accounts (st(|Gas := g'|)) (Address loaded)) = Some (atype.Contract c) ∧
Contract loaded = c)" using contractL addressL 2 a10 * a40 by simp
    have ct1:"∃dud. ep $$ Contract loaded = Some (ct, dud)" using a50 contractL 2 a10 * by blast
    then have fi2:"∀id v. (Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)) = (ct $$ id =
Some (Var v))"
      using a90 unfolding ffold_init_def using ffoldInit_var_storage_mapping_eq[of ct adv c _ v'
loaded] by auto
    then have fi3:"(∃c. Type ((acc) (Address loaded)) = Some (atype.Contract c) ∧ Contract loaded =
c)"
      using fi1 contractL 2(9) transfer_type_same by simp

    have a120:" TypeSafe loaded (acc) (emptyStore) (emptyTypedStore) (Storage st) emptyTypedStore"
using tsInitAcc
      by simp
    moreover have "fullyInitialised loaded (Accounts (st(|Gas := g', Accounts := acc, Stack :=
emptyStore, Memory :=
        emptyTypedStore|))) (Stack (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore|)))"
    proof -
      obtain c_fi where cfi0:"Type (acc (Address loaded)) = Some (atype.Contract c_fi) ∧ Contract
loaded = c_fi"
        using fi3 by blast
      have cfi:"Type (Accounts (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore|)) (Address loaded))
            = Some (atype.Contract c_fi) ∧ Contract loaded = c_fi"
        using cfi0 by simp
      obtain dud_fi where epfi:"ep $$ Contract loaded = Some (ct, dud_fi)"
        using ct1 by blast
      have mapfi:"∀id v. (ct $$ id = Some (Var v)) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id))"
        using fi2 by blast
      have locfi:"∀id v loc. Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
      proof (intro allI impI)
        fix id v loc
        assume h:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc)"
        have hloc:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v,
Storeloc loc) = Storeloc id"
          using tsInitAcc by blast
        then show "id = loc" using h by simp
      qed
      have ptrfi:"∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue loaded) ∧
          accessStore l (Stack (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore|))) = Some (KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue loaded) ∧ CompStoType t' t l' p)"
        by (simp add: accessStore_def emptyStore_def)
      show ?thesis
        unfolding fullyInitialised_def using cfi epfi mapfi locfi ptrfi by blast
    qed
    ultimately have IH1:"StateInvariant loaded (st(|Gas := g', Accounts := acc, Stack := emptyStore,
Memory := emptyTypedStore|)) st'' emptyTypedStore"
      using 7(2)[OF a5 a6 a10 a20 a30 a40 a50 _ _ a60 a70 _ a80 a90 2(8) a110,
          of "(cn, fb')" cn fb' _ _ _ _ _ "(st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory
:= emptyTypedStore|))" st'']
        2(10) 2 * a10 a90 by fastforce

    have accFinal:"Accounts st' = Accounts st''" using 2 by simp
    have storageFinal:"Storage st' = Storage st''" using 2 by simp
```

```
    have acc:"(∀adv. case Type (Accounts st adv) of None ⇒ True | Some EOA ⇒ True | Some
(atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep)"
      using 7(3) unfolding TypeSafe_def AddressTypes_def by blast
    obtain cOld where "Type (Accounts st (Address e)) = Some (atype.Contract cOld) ∧Contract e = cOld"
      using 7(5) unfolding fullyInitialised_def by simp
    then obtain ctO dudO where ctoDef: "ep $$ Contract e = Some (ctO, dudO)" using acc
      using "7.prems"(3) typesafe_base.fullyInitialised_def typesafe_base_axioms by blast
    have SafeC:"safeContract (Accounts st'') (Storage st'')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast

    then have scNew:"∀e ct dud i tp. Type (Accounts
        st'' (Address e)) = Some (atype.Contract (environment.Contract e)) ∧ ep $$ Contract
(e::environment) = Some (ct, dud) ∧ ct
            $$ i = Some (Var tp) ⟶ SCon tp i (Storage st'' (Address e))"
      using SafeC unfolding safeContract_def by simp


    have fi4:"(∀id v. ∃dud. ep $$ Contract loaded = Some (ct, dud)
                          ∧ ct $$ id = Some (Var v) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id)))"
      using fi1 ct1 fi2
      by presburger
    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def using 2
by simp
      next
        show "safeContract (Accounts st') (Storage st')" using IH1 storageFinal 2 unfolding
StateInvariant_def TypeSafe_def by auto
      next
        show "unique_locations (Denvalue e)" using "7.prems"(1) typeSafeUnique by blast
      next
        show "compPointers (Stack st') (Denvalue e)" unfolding compPointers_def
        proof intros
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume a0:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then have a5:"accessStore l1' (Stack st') = accessStore l1' (Stack st)" using 2 by simp
          have a10:"accessStore l2' (Stack st') = accessStore l2' (Stack st)" using 2 by auto
          show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
                else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
            using 7(3) unfolding TypeSafe_def compPointers_def using a0 a5 a10 by simp
        qed
      next
        show "balanceTypes (Accounts st')" using accFinal IH1 unfolding StateInvariant_def
TypeSafe_def by auto
      next
        have "envAddressesWellFormed e" using "7.prems"(1) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using 2 "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs cd" using "7.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using 2 "7.prems"(1) unfolding TypeSafe_def by simp
      next
```

```
        show "typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume tc1:"(t, l) |∈| fmran (Denvalue e)"
          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e)) cd t l"
          proof(cases l)
            case (Stackloc x1)
            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using tc1 Stackloc using 7(3) 2 unfolding TypeSafe_def typeCompat_def
by force
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 2(11) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KCDptr x2)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 2(11) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KMemptr x3)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 7(3) 2(11) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KStoptr x4)
                then have acc0:"accessStore x1 (Stack st) = Some( KStoptr x4)"
                  using Some 2 by auto
                then obtain struct where structDef: "t = type.Storage struct"
                  using 7(3) unfolding TypeSafe_def typeCompat_def using tc1 Some KStoptr Stackloc
                  by (cases t) fastforce+
                have tOld: "Type (Accounts st (Address e)) = Some (atype.Contract (environment.Contract
e))"
                  using "7.prems"(3) unfolding fullyInitialised_def by auto
                obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x4)"
                  using 7(5) using acc0 tc1 unfolding fullyInitialised_def
                  using Stackloc structDef by blast
                then have "SCon tprnt lprnt (Storage st'' (Address e))"
                  using ctoDef IH1 fi4 ct1 scNew
                  by (metis (no_types, lifting) "7.prems"(2,3) tOld
                      accFinal fi_denvalue_storeloc_to_contract_var atype_same)

                then have "SCon struct x4 (Storage st'' (Address e))"
                  using tprntDef SCon_imps_sublocs by blast
                then show ?thesis using Stackloc Some KStoptr structDef storageFinal by simp
              qed
            qed
          next
            case (Storeloc x2)
            then obtain struct where structDef: "t = type.Storage struct"
              using 7(3) tc1 unfolding TypeSafe_def typeCompat_def by (cases t,fastforce+)

            then have "SCon struct x2 (Storage st'' (Address e))"
              using ctoDef IH1 fi4 ct1 scNew fmlookup_ran_iff fullyInitialised_def "7.prems"(3)
            proof -
              have "Type (Accounts st (Address e)) = Some (atype.Contract (environment.Contract e))"
                using "7.prems"(3) unfolding fullyInitialised_def by auto
              then have "Type (Accounts st'' (Address e)) = Some (atype.Contract (environment.Contract
e))"
```

```
            using atype_same[OF "7.prems"(2)] accFinal by simp
          moreover have "ctO $$ x2 = Some (Var struct)"
            using fi_denvalue_storeloc_to_contract_var[OF "7.prems"(3) ctoDef] tc1 Storeloc
structDef by simp
          ultimately show ?thesis using scNew ctoDef by blast
        qed

        then show ?thesis using Storeloc structDef
          by (simp add: storageFinal)
      qed
    qed
  next
    show "denvalueTypeCorrectness e (Stack st') (Memory st')" using 2(11) 7(3) unfolding
TypeSafe_def by simp
  next
    show "subPrefixStructuralConsistency (Memory st')" using 2(11) 7(3) unfolding TypeSafe_def by
simp
  next
    show "SomeValSomeTyp (Memory st')" using 2(11) 7(3) unfolding TypeSafe_def by simp
  qed
next
  have fiOld:"
    ∃c ct dud.
      Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
      Contract e = c ∧
      ep $$ c = Some (ct, dud) ∧
      (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc id)))
∧
      (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
      (∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))"
    using 7(5) unfolding fullyInitialised_def by blast
  then obtain c_fi ct_fi dud_fi where fiC:"
      Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
      Contract e = c_fi ∧
      ep $$ c_fi = Some (ct_fi, dud_fi) ∧
      (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
      (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
    and fiPtr:"∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
    by blast
  have typeNew:"Type (Accounts st' (Address e)) = Some (atype.Contract c_fi)"
    using fiC atype_same "7.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
  have ptrNew:"∀t l p.
        (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
        (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
    using 7(5) 2 unfolding fullyInitialised_def by auto
  show "fullyInitialised e (Accounts st') (Stack st')"
    unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
next
  show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
    using 7(3) 2(11) by simp
next
  show "⋀locs v.
   accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃v'. accessStore locs (Memory st') = Some
(MPointer v')"
```

```
      using 2(11) by simp
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃v'. accessStore locs (Memory
st') = Some (MValue v')"
      using 2(11) by simp
    next
      show "⋀i loc.
       i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹
       accessStore loc (Memory st) = None ⟹ accessStore loc (Memory st') = None"
       using 2(11) by simp
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')"
        using 2(11) by simp
    qed
  qed
next
  case (8 ad ex e cd st)
  show ?case
  proof(cases rule:transfer[OF 8(3)])
    case (1 v t g adv c g' v' acc ct cn f st'')
    then obtain cc :: environment where ccDef: "Contract cc = c"
      by (meson Environment.select_convs(2))
    then have a1:"assert Gas (λst. costs (TRANSFER ad ex) e cd st < state.Gas st) st = Normal ((), st)"
using 1 by simp
    have a10:"modify (λst. st⦇Gas := state.Gas st - costs (TRANSFER ad ex) e cd st⦈) st = Normal ((),
st⦇Gas := state.Gas st - costs (TRANSFER ad ex) e cd st⦈)"
      by simp
    have a20:"toState (local.expr ad e cd) (st⦇Gas := state.Gas st - costs (TRANSFER ad ex) e cd st⦈) =
Normal ((KValue adv, Value TAddr), st⦇Gas := g⦈)"
      using 1 by simp
    then have a30:"(case (KValue adv, Value TAddr) of (KValue adv, Value TAddr) ⇒ return adv | (KValue
adv, Value _)
       ⇒ throw Err | (KValue adv, _) ⇒ throw Err | (_, b) ⇒ throw Err)
     (st⦇Gas := g⦈) = Normal (adv, st⦇Gas := g⦈)" using 1 by simp
    have a40:" toState (local.expr ex e cd) (st⦇Gas := g⦈) = Normal ((KValue v, Value t), st⦇Gas :=
g'⦈)" using 1 by auto
    have a50:"(case (KValue v, Value t) of (KValue v, Value t) ⇒ return (v, t) | (KValue v, _) ⇒
throw Err | (_, b)
       ⇒ throw Err) (st⦇Gas := g'⦈) = Normal ((v, t), (st⦇Gas := g'⦈))"
      using 1 by simp
    then have a60:"option Err (λ_. convert t (TUInt b256) v) (st⦇Gas := g'⦈) = Normal (v', st⦇Gas :=
g'⦈)" using 1 by simp
    have a70:"applyf Accounts (st⦇Gas := g'⦈) = Normal (Accounts (st⦇Gas := g⦈), (st⦇Gas := g'⦈))"
using 1 by simp
    have a80:"Type (Accounts (st⦇Gas := g⦈) adv) = Some (atype.Contract c)" using 1(5) by blast
    have a90:"option Err (λst. transfer (Address e) adv v' (Accounts st)) (st⦇Gas := g'⦈) = Normal
(acc, (st⦇Gas := g'⦈))"
      using 1 by simp
    have a95:"option Err (λ_. ep $$ c) (st⦇Gas := g'⦈) = Normal ((ct, cn, f), (st⦇Gas := g'⦈))" using
1 by auto
    have a100:"modify (λst. st⦇Accounts := acc, Stack := emptyStore, Memory := emptyTypedStore⦈)
(st⦇Gas := g'⦈)
                  = Normal ((), st⦇Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore⦈)" by simp
    obtain loaded where loadedDef:"loaded = ffold_init ct (emptyEnv adv c (Address e) v') (fmdom ct)"
by blast

    have tsLoaded':"∀x y. TypeSafe loaded acc emptyStore emptyTypedStore (Storage st) emptyTypedStore
∧
      (Denvalue loaded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue loaded $$ x = Some y ⟶
          (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    proof -
      have "safeContract (Accounts st) (Storage st)" using 8(2) unfolding TypeSafe_def AddressTypes_def
using 1 by simp
```

```
      moreover have "safeContract acc (Storage st)"
        using "1"(7) calculation safeContract_def transfer_type_same by force
      moreover have svv':"svalueTypes v'" unfolding svalueTypes_def
        using 1(4,3) 8(2) 8(4)
        using exprTypeconInduct(3)[of ex e cd "(st⦇Gas := g⦈)" g "KValue v" "Value t" g']
          typeSafeConvert[of t v "(TUInt b256)"] convertSame by auto
      moreover have "balanceTypes acc" unfolding balanceTypes_def
      proof -
        have old:"balanceTypes (Accounts (st⦇Gas := g'⦈))" using 8(2) unfolding TypeSafe_def by simp
        have accExp:"(case subBalance (Address e) v' (Accounts (st)) of None ⇒ None | Some x ⇒
addBalance adv v' x) = Some acc"
          using 1(7) unfolding transfer_def by blast
        have v'TC: "typeCon (TUInt b256) v'"
          using svv' unfolding svalueTypes_def by blast

        show "∀adv'. typeCon (TUInt b256) (Bal (acc adv')) "
        proof(cases "adv = (Address e)")
          case same:True
          show ?thesis using transfer_sameRead
            by (metis "1"(7) "8"(2) balanceTypes_def same typeSafeAccounts transfer_eq v'TC)
        next
          case notSame:False
          show ?thesis
          proof
            fix adv'
            show "typeCon (TUInt b256) (Bal (acc adv')) "
            proof(cases "adv' = (Address e)")
              case True
              then have tco:"typeCon (TUInt b256) (Bal ((Accounts (st⦇Gas := g'⦈)) adv'))"
                using True old accExp balanceTypes_def by blast
              have "Bal (acc (Address e)) = ShowL_int (ReadL_int (Bal (Accounts (st⦇Gas :=
                  g'⦈) (Address e))) - (ReadL_int v'))"
                using transfer_subRead[OF 1(7) _ v'TC ] True notSame tco by auto
              then show ?thesis using True
                using "1"(7,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈)
adv'))› checkUInt_def transfer_val1 transfer_val2
              proof -
                have "∀a. typeCon (TUInt b256) (Bal (Accounts st a))"
                  using old unfolding balanceTypes_def by simp
                then show ?thesis using transfer_Bal_typeCon[unfolded transfer_def, OF accExp] by
blast
              qed
            next
              case False
              then show ?thesis
              proof(cases "adv' = adv")
                case True
                then have "typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈) adv))"
                  using True old accExp balanceTypes_def by simp
                then have "Bal (acc adv) = (ShowL_int ((ReadL_int (Bal (Accounts (st⦇Gas := g'⦈) adv))) +
(ReadL_int v')))"
                  using transfer_addRead[OF 1(7)_ v'TC ] 1(3) notSame by auto
                then show ?thesis using True
                  using "1"(7,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈)
adv))› checkUInt_def transfer_val1 transfer_val2 notSame by auto
              next
                case f2:False
                then show ?thesis using False transfer_eq[OF 1(7)] old
                  using balanceTypes_def by force
              qed
            qed
          qed
        qed
      qed
```

```
      moreover have "lessThanTopLocs emptyStore" using emptyTopLocs by simp
      moreover have "ep $$ Contract cc = Some (ct, cn, f)" using ccDef 1(6) by simp
      moreover have "addressFormat adv" using 8(2) unfolding TypeSafe_def AddressTypes_def using 1 by
(simp split:option.splits atype.splits)
      moreover have "addressFormat (Sender e)" using 8(2) unfolding TypeSafe_def AddressTypes_def
using 1 by simp
      moreover have "AddressTypes acc" using 1(7) transfer_type_same 8(2) unfolding TypeSafe_def un-
folding AddressTypes_def by simp
      moreover have "addressFormat (Address e)" using "8.prems"(1) unfolding TypeSafe_def by simp
      moreover have "SomeValSomeTyp emptyTypedStore" unfolding SomeValSomeTyp_def accessTypeStore_def
accessStore_def emptyTypedStore_def by auto
      moreover have sp:"subPrefixStructuralConsistency emptyTypedStore" unfolding
subPrefixStructuralConsistency_def accessTypeStore_def emptyTypedStore_def by simp
      moreover have "Type (acc adv) = Some (atype.Contract (Contract cc))"
        using a80 ccDef transfer_type_same[OF 1(7)] by simp
      ultimately show ?thesis
        using loadedDef ccDef unfolding ffold_init_def
        using ffoldInitTypeSafe[of acc "Storage st" v' emptyTypedStore cc ct "(cn,f)" adv "(Address e)"
" (fmdom ct)"]
        using typedEmptyTopLocs
        by blast
    qed
    then have tsLoaded:"TypeSafe loaded (Accounts (st(|Gas := g', Accounts := acc, Stack := emptyStore,
Memory := emptyTypedStore|)))
     (Stack (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory := emptyTypedStore|))) (Memory
(st(|Gas := g',
         Accounts := acc, Stack := emptyStore, Memory := emptyTypedStore|)))
     (Storage (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory := emptyTypedStore|)))
emptyTypedStore" by auto

    have contractL:"Contract loaded = c" using loadedDef by simp
    have addressL:"Address loaded = adv" using loadedDef by simp
    have fi1:"(∃c. Type (Accounts (st(|Gas := g'|)) (Address loaded)) = Some (atype.Contract c) ∧
Contract loaded = c)" using contractL addressL 1 a10 a40 by simp
    have ctd:"∃dud. ep $$ Contract loaded = Some (ct, dud)" using a50 contractL 1 a10 by blast
    then have fi2:"∀id v. (Denvalue loaded $$ id = Some (type.Storage v, Storeloc id)) = (ct $$ id =
Some (Var v))"
      using loadedDef unfolding ffold_init_def using ffoldInit_var_storage_mapping_eq[of ct adv c
"(Address e)" v' loaded] by auto
    then have fi3:"(∃c. Type ((acc) (Address loaded)) = Some (atype.Contract c) ∧ Contract loaded =
c)"
      using fi1 contractL 1(7) transfer_type_same by simp
    have fi4:"(∀id v. ∃dud. ep $$ Contract loaded = Some (ct, dud)
                          ∧ ct $$ id = Some (Var v) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id)))"
      using fi1 ctd fi2
      by (metis (mono_tags, opaque_lifting))
    have fiLoaded:"fullyInitialised loaded (Accounts (st(|Gas := g', Accounts := acc, Stack :=
emptyStore, Memory :=
       emptyTypedStore|))) (Stack (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore|)))"
    proof -
      obtain c_fi where cfi0:"Type (acc (Address loaded)) = Some (atype.Contract c_fi) ∧ Contract
loaded = c_fi"
        using fi3 by blast
      have cfi:"Type (Accounts (st(|Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore|)) (Address loaded))
           = Some (atype.Contract c_fi) ∧ Contract loaded = c_fi"
        using cfi0 by simp
      obtain dud_fi where epfi:"ep $$ Contract loaded = Some (ct, dud_fi)"
        using ctd by blast
      have mapfi:"∀id v. (ct $$ id = Some (Var v)) = (Denvalue loaded $$ id = Some (type.Storage v,
Storeloc id))"
        using fi2 by blast
```

```
      have locfi:"∀id v loc. Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
      proof (intro allI impI)
        fix id v loc
        assume h:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc)"
        have hloc:"Denvalue loaded $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v,
Storeloc loc) = Storeloc id"
          using tsLoaded' by blast
        then show "id = loc" using h by simp
      qed
      have ptrfi:"∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue loaded) ∧
          accessStore l (Stack (st⦇Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore⦈)) = Some (KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue loaded) ∧ CompStoType t' t l' p)"
        by (simp add: accessStore_def emptyStore_def)
      show ?thesis
        unfolding fullyInitialised_def using cfi epfi mapfi locfi ptrfi by blast
    qed
    have stmtC:"stmt f loaded emptyTypedStore (st⦇Gas := g', Accounts := acc, Stack := emptyStore,
Memory := emptyTypedStore⦈) = Normal ((), st'')"
      using 1(8) loadedDef unfolding ffold_init_def by simp
    have IH1:"StateInvariant loaded (st⦇Gas := g', Accounts := acc, Stack := emptyStore, Memory :=
emptyTypedStore⦈) st'' emptyTypedStore"
      using 8(1)[OF a1 a10 a20 a30 a40 a50 _ a60 a70 a80 _ a95 _ _ loadedDef _ _ a90 a100 tsLoaded
stmtC fiLoaded ] by simp

    have acc:"(∀adv. case Type (Accounts st adv) of None ⇒ True | Some EOA ⇒ True | Some
(atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep)"
      using 8(2) unfolding TypeSafe_def AddressTypes_def by blast
    obtain cOld where "Type (Accounts st (Address e)) = Some (atype.Contract cOld) ∧ Contract e =
cOld"
      using 8(4) unfolding fullyInitialised_def by simp
    then obtain ctO dudO where ctoDef: "ep $$ Contract e = Some (ctO, dudO)" using acc
      using "8.prems"(3) typesafe_base.fullyInitialised_def typesafe_base_axioms by blast
    have storageFinal:"Storage st' = Storage st''" using 1 by auto

    have ct1:"∃dud. ep $$ Contract loaded = Some (ct, dud)" using a50 contractL 1 a10 by blast
    have SafeC:"safeContract (Accounts st'') (Storage st'')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast

    then have scNew:"∀e ct dud i tp. Type (Accounts st'' (Address e)) = Some (atype.Contract
(environment.Contract e))
                    ∧ ep $$ Contract (e::environment) = Some (ct, dud) ∧ ct $$ i = Some (Var tp) ⟶
SCon tp i (Storage st'' (Address e))"
      using SafeC unfolding safeContract_def by simp
    have accFinal:"Accounts st' = Accounts st''" using 1 by simp

    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def using 1
by simp
      next
        show "safeContract (Accounts st') (Storage st')" using IH1 1 unfolding StateInvariant_def
TypeSafe_def by auto
      next
        show "unique_locations (Denvalue e)" using "8.prems"(1) typeSafeUnique by blast
      next
        have "compPointers (Stack st'') (Denvalue loaded)"
          using IH1 unfolding StateInvariant_def TypeSafe_def by blast
        show "compPointers (Stack st') (Denvalue e)" unfolding compPointers_def
        proof intros
```

```
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume a0:"(type.Storage tp1, l1) |∈| fmran (Denvalue e) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then have a5:"accessStore l1' (Stack st') = accessStore l1' (Stack st)" using 1 by simp
          have a10:"accessStore l2' (Stack st') = accessStore l2' (Stack st)" using 1 by auto
          show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
                else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
            using 8(2) unfolding TypeSafe_def compPointers_def using a0 a5 a10 by simp
        qed
      next
        show "balanceTypes (Accounts st')" using 1 IH1 unfolding StateInvariant_def TypeSafe_def by
auto
      next
        have "envAddressesWellFormed e" using "8.prems"(1) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using 1 "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs cd" using "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using 1 "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume tc1:"(t, l) |∈| fmran (Denvalue e)"
          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e)) cd t l"
          proof(cases l)
            case (Stackloc x1)
            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using tc1 Stackloc using 8(2) 1 unfolding TypeSafe_def typeCompat_def
by force
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 8(2) 1(9) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KCDptr x2)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 8(2) 1(9) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KMemptr x3)
                then show ?thesis apply(cases t) using tc1 Stackloc Some using 8(2) 1(9) unfolding
TypeSafe_def typeCompat_def by force+
              next
                case (KStoptr x4)
                then have acc0:"accessStore x1 (Stack st) = Some( KStoptr x4)"
                  using Some 1 KStoptr by auto
                then obtain struct where structDef: "t = type.Storage struct"
                  using 8(2) unfolding TypeSafe_def typeCompat_def using tc1 Some KStoptr Stackloc
                  by (cases t) fastforce+
                have tOld: "Type (Accounts st (Address e)) = Some (atype.Contract (environment.Contract
e))"
                  using "8.prems"(3) unfolding fullyInitialised_def by auto
```

```
              obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x4)"
                using 8(4) using acc0 tc1 unfolding fullyInitialised_def
                using Stackloc structDef by blast

              then have "SCon tprnt lprnt (Storage st'' (Address e))"
                using ctoDef IH1 fi4 ct1 scNew
                  "8.prems"(2,3) tOld
                    fi_denvalue_storeloc_to_contract_var atype_same accFinal
                by (metis (no_types, lifting))

              then have "SCon struct x4 (Storage st'' (Address e))"
                using tprntDef SCon_imps_sublocs by blast
              then show ?thesis using Stackloc Some KStoptr structDef storageFinal by simp
            qed
          qed
        next
          case (Storeloc x2)
          then obtain struct where structDef: "t = type.Storage struct"
            using 8(2) tc1 unfolding TypeSafe_def typeCompat_def by (cases t,fastforce+)

          then have "SCon struct x2 (Storage st'' (Address e))"
            using ctoDef IH1 fi4 ct1 scNew fmlookup_ran_iff fullyInitialised_def "8.prems"(3)
          proof -
            have "Type (Accounts st (Address e)) = Some (atype.Contract (environment.Contract e))"
              using "8.prems"(3) unfolding fullyInitialised_def by auto
            then have "Type (Accounts st'' (Address e)) = Some (atype.Contract (environment.Contract
e))"
              using atype_same[OF "8.prems"(2)] accFinal by simp
            moreover have "ctO $$ x2 = Some (Var struct)"
              using fi_denvalue_storeloc_to_contract_var[OF "8.prems"(3) ctoDef] tc1 Storeloc
structDef by simp
            ultimately show ?thesis using scNew ctoDef by blast
          qed

          then show ?thesis using Storeloc structDef
            by (simp add: storageFinal)
        qed
      qed
    next
      show "denvalueTypeCorrectness e (Stack st') (Memory st')" using 1 8(2) unfolding TypeSafe_def
by simp
    next
      show "subPrefixStructuralConsistency (Memory st')" using 1 8(2) unfolding TypeSafe_def by simp
    next
      show "SomeValSomeTyp (Memory st')" using 1 8(2) unfolding TypeSafe_def by simp
    qed
  next
    have fiOld:"
      ∃c ct dud.
        Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
        Contract e = c ∧
        ep $$ c = Some (ct, dud) ∧
        (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc id)))
∧
        (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
        (∀t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
            (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))"
      using 8(4) unfolding fullyInitialised_def by blast
    then obtain c_fi ct_fi dud_fi where fiC:"
        Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
        Contract e = c_fi ∧
```

```
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
        and fiPtr:"∀ t l p.
             (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
             (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
        by blast
      have typeNew:"Type (Accounts st' (Address e)) = Some (atype.Contract c_fi)"
        using fiC atype_same "8.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
      have ptrNew:"∀ t l p.
           (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
           (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
        using 8(4) 1 unfolding fullyInitialised_def by auto
      show "fullyInitialised e (Accounts st') (Stack st')"
        unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
    next
      show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
        using 8(3) 1(9) by simp
    next
      show "⋀locs v.
       accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory st') = Some
(MPointer v')"
        using 1(9) by simp
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
        using 1(9) by simp
    next
      show "⋀i loc.
       i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹
       accessStore loc (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using 1(9) by simp
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')" using 1(9) by simp

    qed
  next
    case (2 v t g adv g' v' acc)
    then have a1:"assert Gas (λst. costs (TRANSFER ad ex) e cd st < state.Gas st) st = Normal ((), st)"
by simp
    have a10:"modify (λst. st(|Gas := state.Gas st - costs (TRANSFER ad ex) e cd st|)) st = Normal ((),
st(|Gas := state.Gas st - costs (TRANSFER ad ex) e cd st|))"
      by simp
    have a20:"toState (local.expr ad e cd) (st(|Gas := state.Gas st - costs (TRANSFER ad ex) e cd st|)) =
Normal ((KValue adv, Value TAddr), st(|Gas := g|))"
      using 2 by simp
    then have a30:"(case (KValue adv, Value TAddr) of (KValue adv, Value TAddr) ⇒ return adv | (KValue
adv, Value _)
        ⇒ throw Err | (KValue adv, _) ⇒ throw Err | (_, b) ⇒ throw Err)
     (st(|Gas := g|)) = Normal (adv, st(|Gas := g|))" using 2 by simp
    have a40:" toState (local.expr ex e cd) (st(|Gas := g|)) = Normal ((KValue v, Value t), st(|Gas :=
g'|))" using 2 by auto
    have a50:"(case (KValue v, Value t) of (KValue v, Value t) ⇒ return (v, t) | (KValue v, _) ⇒
throw Err | (_, b)
        ⇒ throw Err) (st(|Gas := g'|)) = Normal ((v, t), (st(|Gas := g'|)))"
      using 2 by simp
    then have a60:"option Err (λ_. convert t (TUInt b256) v) (st(|Gas := g'|)) = Normal (v', st(|Gas :=
g'|))" using 2 by simp
    have a70:"applyf Accounts (st(|Gas := g'|)) = Normal (Accounts (st(|Gas := g|)), (st(|Gas := g'|)))"
```

```
using 2 by simp
    have a90:"option Err (λst. transfer (Address e) adv v' (Accounts st)) (st⦇Gas := g'⦈) = Normal
(acc, (st⦇Gas := g'⦈))"
      using 2 by simp

    have svv':"svalueTypes v'" unfolding svalueTypes_def
      using 2(4,3) 8(2) 8(4)
      using exprTypeconInduct(3)[of ex e cd "(st⦇Gas := g⦈)" g "KValue v" "Value t" g']
        typeSafeConvert[of t v "(TUInt b256)"] convertSame by auto

    have btACC:"balanceTypes acc" unfolding balanceTypes_def
    proof -
      have old:"balanceTypes (Accounts (st⦇Gas := g'⦈))" using 8(2) unfolding TypeSafe_def by simp
      have accExp:"(case subBalance (Address e) v' (Accounts (st)) of None ⇒ None | Some x ⇒
addBalance adv v' x) = Some acc"
        using 2(6) unfolding transfer_def by blast
      have v'TC: "typeCon (TUInt b256) v'"
        using svv' unfolding svalueTypes_def by blast

      show "∀adv'. typeCon (TUInt b256) (Bal (acc adv')) "
      proof(cases "adv = (Address e)")
        case same:True
        show ?thesis using transfer_sameRead
          by (metis "2"(6) "8"(2) balanceTypes_def same typeSafeAccounts transfer_eq v'TC)
      next
        case notSame:False
        show ?thesis
        proof
          fix adv'
          show "typeCon (TUInt b256) (Bal (acc adv')) "
          proof(cases "adv' = (Address e)")
            case True
            then have tco:"typeCon (TUInt b256) (Bal ((Accounts (st⦇Gas := g'⦈)) adv'))"
              using True old accExp balanceTypes_def by blast
            have "Bal (acc (Address e)) = ShowL_int (ReadL_int (Bal (Accounts (st⦇Gas := g'⦈)) (Address
e))) - (ReadL_int v'))"
              using transfer_subRead[OF 2(6) _ v'TC ] True notSame tco by auto
            then show ?thesis using True
              using "2"(6,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈)) adv'))›
checkUInt_def transfer_val1 transfer_val2
            proof -
              have "∀a. typeCon (TUInt b256) (Bal (Accounts st a))"
                using old unfolding balanceTypes_def by simp
              then show ?thesis using transfer_Bal_typeCon[unfolded transfer_def, OF accExp] by blast
            qed
          next
            case False
            then show ?thesis
            proof(cases "adv' = adv")
              case True
              then have "typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈)) adv))"
                using True old accExp balanceTypes_def by simp
              then have "Bal (acc adv) = (ShowL_int ((ReadL_int (Bal (Accounts (st⦇Gas := g'⦈)) adv))) +
(ReadL_int v'))"
                using transfer_addRead[OF 2(6)_ v'TC ] 2(3) notSame by auto
              then show ?thesis using True
                using "2"(6,3) Read_ShowL_id ‹typeCon (TUInt b256) (Bal (Accounts (st⦇Gas := g'⦈))
adv))› checkUInt_def transfer_val1 transfer_val2 notSame by auto
            next
              case f2:False
              then show ?thesis using False transfer_eq[OF 2(6)] old
                using balanceTypes_def by force
            qed
          qed
```

```
        qed
      qed
    qed

    have accfinal:"Accounts st'= acc" using 2 by simp

    have sameStores:"Storage st' = Storage st ∧ Memory st = Memory st' ∧ Stack st = Stack st'" using 2
by simp
    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" unfolding AddressTypes_def
        proof intros
          fix adv
          show "case Type (Accounts st' adv) of None ⇒ True | Some EOA ⇒ True | Some (atype.Contract
c) ⇒ addressFormat adv ∧ c |∈| fmdom ep"
          proof (cases "Type (Accounts st' adv)")
            case None
            then show ?thesis by simp
          next
            case (Some a)
            then show ?thesis
            proof(cases a)
              case EOA
              then show ?thesis using Some by simp
            next
              case (Contract x2)
              then show ?thesis using Some 8(2) unfolding TypeSafe_def using accfinal 2(6)
transfer_type_same
                by (metis transfer_type_same accfinal Some "2"(6) AddressTypes_def)
            qed
          qed
        qed
      next
        show "safeContract (Accounts st') (Storage st')" using 8(2) 2 unfolding TypeSafe_def using
sameStores
        proof -
          have "safeContract (Accounts st) (Storage st')" using 8(2) sameStores unfolding TypeSafe_def
by simp
          moreover have "∀a. Type (Accounts st' a) = Type (Accounts st a)"
            using transfer_type_same[OF "2"(6)] accfinal by simp
          ultimately show ?thesis by (rule safeContract_Type_cong)
        qed
      next
        show "unique_locations (Denvalue e)" using 8(2) unfolding TypeSafe_def using sameStores by
simp
      next
        show "compPointers (Stack st') (Denvalue e)" using 8(2) unfolding TypeSafe_def using
sameStores by simp
      next
        show "balanceTypes (Accounts st')" using btACC accfinal by simp
      next
        have "envAddressesWellFormed e" using "8.prems"(1) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using 8(2) unfolding TypeSafe_def using sameStores by simp
      next
        show "lessThanTopLocs cd" using "8.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using 8(2) unfolding TypeSafe_def using sameStores by
```

```
simp
      next
        show "typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
          using 8(2) unfolding TypeSafe_def using sameStores by simp
      next
        show "denvalueTypeCorrectness e (Stack st') (Memory st')" using 8(2) unfolding TypeSafe_def
using sameStores by simp
      next
        show "subPrefixStructuralConsistency (Memory st')" using 8(2) unfolding TypeSafe_def using
sameStores by simp
      next
        show "SomeValSomeTyp (Memory st')" using 8(2) unfolding TypeSafe_def using sameStores by simp
      qed
    next
      have fiOld:"
             ∃ c ct dud.
               Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
               Contract e = c ∧
               ep $$ c = Some (ct, dud) ∧
               (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
               (∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
               (∀ t l p.
                   (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) =
Some (KStoptr p) ⟶
                   (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t
l' p))"
        using 8(4) unfolding fullyInitialised_def by blast
      then obtain c_fi ct_fi dud_fi where fiC:"
          Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
          Contract e = c_fi ∧
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
        and fiPtr:"∀ t l p.
             (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
             (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
        by blast
      have typeNew:"Type (Accounts st' (Address e)) = Some (atype.Contract c_fi)"
        using fiC atype_same "8.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
      have ptrNew:"∀ t l p.
           (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
           (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
        using 8(4) 2 unfolding fullyInitialised_def by auto
      show "fullyInitialised e (Accounts st') (Stack st')"
        unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
    next
      show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
        using 8(3) 2(7) by simp
    next
      show "⋀locs v.
       accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory st') = Some
(MPointer v')"
        using 2(7) by simp
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
        using 2(7) by simp
    next
```

```
      show "⋀i loc.
       i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹
       accessStore loc (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using 2(7) by simp
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')" using 2(7) by simp
    qed
  qed
next
  case (9 id0 tp s e_v cd st)
  show ?case
  proof(cases rule:blockNone[OF 9(3)])
    case (1 cd' mem' sck' e')
    have a10:"assert Gas (λst. costs (BLOCK (id0, tp, None) s) e_v cd st < state.Gas st) st = Normal
((), st)" using 1 by simp
    obtain sgas where sgasDef:"sgas = state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st" by simp
    then have a20:"modify (λst. st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|)) st
= Normal ((), (st(|Gas := sgas|) ))" by simp
    have a30:"option Err (λst. decl id0 tp None False cd (Memory st) (Storage st (Address e_v)) (cd,
Memory st, Stack st, e_v)) (st(|Gas := sgas|) )
           = Normal ((cd', mem', sck', e'), (st(|Gas := sgas|) )) " using 1 by simp
    have a40:"modify (λst. st(|Stack := sck', Memory := mem'|)) (st(|Gas := sgas|)) = Normal ((), st(|Gas
:= sgas, Stack := sck', Memory := mem'|))"
      by simp

    have tsDecl:"TypeSafe e' (Accounts (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd
st|))) sck' mem'
     (Storage (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|))) cd' ∧
    (∀x l. (∄y y'. x = type.Memory y ∨ x = Value y') ∧ (x, l) |∈| fmran (Denvalue e') ⟶
          (x, l) |∈| fmran (Denvalue e_v)) ∧
    (∀sckl ptr.
        accessStore sckl sck' = Some ptr ∧ (∄y y'. ptr = KMemptr y ∨ ptr = KValue y') ⟶
        accessStore sckl (Stack (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|))) =
        Some ptr) ∧
    cd = cd'∧
    Toploc (Memory (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|))) ≤ Toploc mem'
∧
    (∀locs v.
        accessStore locs (Memory (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|)))
= Some v ⟶
        accessStore locs mem' = Some v) ∧
    (∀locs t.
        accessTypeStore locs (Memory (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd
st|))) = Some t ⟶
        accessTypeStore locs mem' = Some t) ∧
    (∀locs.
        (∃tloc<Toploc (Memory (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd st|))).
            LSubPrefL2 locs (ShowL_nat tloc) ∧
            accessStore locs (Memory (st(|Gas := state.Gas st - costs (BLOCK (id0, tp, None) s) e_v cd
st|))) = None) ⟶
        accessStore locs mem' = None)"
      using typeSafeDeclNone[OF _ 1(2)] 9(2) by simp

    then have a50:"TypeSafe e' (Accounts (st(|Gas := sgas, Stack := sck', Memory := mem'|))) (Stack
(st(|Gas := sgas, Stack := sck', Memory := mem'|)))
   (Memory (st(|Gas := sgas, Stack := sck', Memory := mem'|))) (Storage (st(|Gas := sgas, Stack := sck',
Memory := mem'|))) cd'" by simp
    have denLink:"∀ii x. Denvalue e_v $$ ii = Some x ⟶ Denvalue e' $$ ii = Some x" using
decl_env_monotonic[OF 1(2)] by auto

    have fiDec:" fullyInitialised e' (Accounts (st(|Gas := sgas, Stack := sck', Memory := mem'|))) (Stack
(st(|Gas := sgas, Stack := sck', Memory := mem'|)))"
    proof -
      have sameStuff:"Address e' = Address e_v ∧ Contract e' = Contract e_v"
```

```
      using 1(2) decl_env by simp
    have fiOld:"
      ∃c ct dud.
        Type (Accounts st (Address e_v)) = Some (atype.Contract c) ∧
        Contract e_v = c ∧
        ep $$ c = Some (ct, dud) ∧
        (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)))
∧
        (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
        (∀t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
            (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p))"
      using 9(4) unfolding fullyInitialised_def by blast
    then obtain c_fi ct_fi dud_fi where fiC:"
        Type (Accounts st (Address e_v)) = Some (atype.Contract c_fi) ∧
        Contract e_v = c_fi ∧
        ep $$ c_fi = Some (ct_fi, dud_fi) ∧
        (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc
id))) ∧
        (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
      and fiPtr:"∀t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
            (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p)"
      by blast

    have cfi:"Type (Accounts (st(|Gas := sgas, Stack := sck', Memory := mem'|)) (Address e')) = Some
(atype.Contract c_fi) ∧
        Contract e' = c_fi ∧ ep $$ c_fi = Some (ct_fi, dud_fi)"
      using fiC sameStuff by simp

    have mapfi:"∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e' $$ id = Some (type.Storage v,
Storeloc id))"
    proof (intro allI)
      fix id v
      show "(ct_fi $$ id = Some (Var v)) = (Denvalue e' $$ id = Some (type.Storage v, Storeloc id))"
      proof
        assume hct:"ct_fi $$ id = Some (Var v)"
        then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)"
          using fiC by blast
        then show "Denvalue e' $$ id = Some (type.Storage v, Storeloc id)"
          using decl_env_monotonic[OF 1(2)] by blast
      next
        assume hnew:"Denvalue e' $$ id = Some (type.Storage v, Storeloc id)"
        show "ct_fi $$ id = Some (Var v)"
        proof (cases "id = id0")
          case True
          have "∀t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
            using decl_env_storlocs_unchanged[OF 1(2)] by simp
          then have "Denvalue e_v $$ id0 = Some (type.Storage v, Storeloc id0)"
            using hnew True by blast
          then show ?thesis using fiC True by blast
        next
          case False
          then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)"
            using decl_env_not_i[OF 1(2)] hnew by blast
          then show ?thesis using fiC by blast
        qed
      qed
    qed
```

```
      have locfi:"∀id v loc. Denvalue e' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
      proof (intro allI impI)
        fix id v loc
        assume hnew:"Denvalue e' $$ id = Some (type.Storage v, Storeloc loc)"
        show "id = loc"
        proof (cases "id = id0")
          case True
          have "∀t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
            using decl_env_storlocs_unchanged[OF 1(2)] by simp
          then have "Denvalue e_v $$ id0 = Some (type.Storage v, Storeloc loc)"
            using hnew True by blast
          then have "id0 = loc" using fiC by blast
          then show ?thesis using True by simp
        next
          case False
          then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc)"
            using decl_env_not_i[OF 1(2)] hnew by blast
          then show ?thesis using fiC by blast
        qed
      qed

      have ptrfi:"∀t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e') ∧
          accessStore l (Stack (st(|Gas := sgas, Stack := sck', Memory := mem'|))) = Some (KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e') ∧ CompStoType t' t l' p)"
      proof (intro allI impI)
        fix t l p
        assume in1:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e') ∧
            accessStore l (Stack (st(|Gas := sgas, Stack := sck', Memory := mem'|))) = Some (KStoptr p)"
        have inOld:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v)" using tsDecl in1 by simp
        have accOld:"accessStore l (Stack st) = Some (KStoptr p)" using tsDecl in1 by simp
        from fiPtr inOld accOld obtain t' l' where oldRan:
            "(type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v)"
            and comp:"CompStoType t' t l' p"
          by blast
        from oldRan obtain idp where idpDef:"Denvalue e_v $$ idp = Some (type.Storage t', Storeloc
l')"
          using fmlookup_ran_iff by fast
        have "Denvalue e' $$ idp = Some (type.Storage t', Storeloc l')"
          using decl_env_monotonic[OF 1(2)] idpDef by blast
        then have newRan:"(type.Storage t', Storeloc l') |∈| fmran (Denvalue e')"
          by (meson fmranI)
        show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e') ∧ CompStoType t' t l' p"
          using newRan comp by blast
      qed

      show ?thesis
        unfolding fullyInitialised_def using cfi mapfi locfi ptrfi by blast
    qed

    have IH1:"StateInvariant e' (st(|Gas := sgas, Stack := sck', Memory := mem'|)) st' cd'"
      using 9(1)[OF a10 a20 a30, of cd' "(mem', sck', e')" mem' "(sck', e')" sck' e', OF _ _ _ a40 a50
_ fiDec, of st'] 1(3) sgasDef by simp
    then have fiFinal:" fullyInitialised e' (Accounts st') (Stack st')"
      unfolding StateInvariant_def by blast
    have sameAdd:"Address e' = Address e_v" using 1(2) decl_env by simp
    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e_v (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
```

```
blast
      next
        show "safeContract (Accounts st') (Storage st')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast
      next
        show "unique_locations (Denvalue e_v)" using 9(2) unfolding TypeSafe_def by simp
      next
        have cpNew:"(∀ tp1 tp2 l1 l2 l1' l2' stl1 stl2.
       (type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e') ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2) ⟶
       (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if TypedStoSubpref stl1
stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
          using IH1 unfolding StateInvariant_def TypeSafe_def compPointers_def by auto
        show "compPointers (Stack st') (Denvalue e_v)" unfolding compPointers_def
        proof intros
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume a0:"(type.Storage tp1, l1) |∈| fmran (Denvalue e_v) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e_v) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then obtain i1 i2 where iDefs:" (Denvalue e_v) $$ i1 = Some (type.Storage tp1, l1)
                                  ∧ (Denvalue e_v) $$ i2 = Some (type.Storage tp2, l2)" by blast
          then have "(type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧ (type.Storage tp2, l2) |∈| fmran
(Denvalue e')" using a0
            by (metis denLink fmlookup_ran_iff)
          then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
                else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
            using cpNew using a0 by blast
        qed
      next
        show "balanceTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        have "envAddressesWellFormed e_v" using 9(2) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e_v)" and "addressFormat (Sender e_v)" by simp+
      next
        show "svalueTypes (Svalue e_v)" using 9(2) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        show "lessThanTopLocs cd" using 9(2) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        have oldTC:"typeCompat (Denvalue e') (Stack st') (Memory st') (Storage st' (Address e')) cd'"
          using IH1 unfolding StateInvariant_def TypeSafe_def by blast
        show "typeCompat (Denvalue e_v) (Stack st') (Memory st') (Storage st' (Address e_v)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume inDen:"(t, l) |∈| fmran (Denvalue e_v)"
          then have inDenN:"(t, l) |∈| fmran (Denvalue e')"
            using denLink by (meson fmlookup_ran_iff)

          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e_v)) cd t l"
          proof(cases l)
            case (Stackloc x1)
            then have oldTC':"(
             case accessStore x1 (Stack st') of None ⇒ False
              | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
```

```
              | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd' stloc | _ ⇒ False)
              | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc
| _ ⇒ False)
              | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address e')) | _ ⇒ False))"
              using oldTC unfolding typeCompat_def using inDenN by fastforce
            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Stackloc by
fastforce
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1')
                then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Some
Stackloc by force
              next
                case (KCDptr x2)
                then obtain struct where structDef:"t = Calldata struct ∧ MCon struct cd' x2"
                  using oldTC' unfolding typeCompat_def using inDenN sameAdd Some Stackloc by (cases
t; auto)
                then show ?thesis using tsDecl KCDptr Some Stackloc by simp
              next
                case (KMemptr x3)
                then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Some
Stackloc by fastforce
              next
                case (KStoptr x4)
                then have "case t of type.Storage struct ⇒ SCon struct x4 (Storage st' (Address e')) |
_ ⇒ False"
                  using oldTC' using inDenN sameAdd Some Stackloc by (auto split:option.splits
type.splits stackvalue.splits denvalue.splits)
                then obtain struct where "t = type.Storage struct ∧ SCon struct x4 (Storage st'
(Address e'))" by (cases t; auto)
                then show ?thesis using Some Stackloc KStoptr
                  by (simp add: sameAdd)
              qed
            qed
          next
            case (Storeloc x2)
            then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd
              by (metis (no_types, lifting) denvalue.simps(6) type.exhaust type.simps(17,18,19,20))
          qed
        qed
      next
        show "denvalueTypeCorrectness e_v (Stack st') (Memory st')"
          unfolding denvalueTypeCorrectness_def
        proof intros
          fix t l ptr_loc sub_loc
          assume in1:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st')
= Some (KMemptr ptr_loc)"
          then have in2:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e')" using denLink
            by (meson fmlookup_ran_iff)
          then show " case t of MTArray len arr ⇒ ∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i))
(Memory st') = Some arr
       | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val)"
            using IH1 unfolding StateInvariant_def TypeSafe_def denvalueTypeCorrectness_def
            using in1 in2 by blast
        qed
      next
        show "subPrefixStructuralConsistency (Memory st')" using IH1 unfolding StateInvariant_def
TypeSafe_def by simp
```

```
      next
        show "SomeValSomeTyp (Memory st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
simp
      qed
    next
      have fiOld:"
        ∃c ct dud.
          Type (Accounts st (Address e_v)) = Some (atype.Contract c) ∧
          Contract e_v = c ∧
          ep $$ c = Some (ct, dud) ∧
          (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)))
∧
          (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
          (∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p))"
        using 9(4) unfolding fullyInitialised_def by blast
      then obtain c_fi ct_fi dud_fi where fiC:"
          Type (Accounts st (Address e_v)) = Some (atype.Contract c_fi) ∧
          Contract e_v = c_fi ∧
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
        and fiPtr:"∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p)"
        by blast
      have typeNew:"Type (Accounts st' (Address e_v)) = Some (atype.Contract c_fi)"
        using fiC atype_same "9.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
      have ptrNew:"∀t l p.
           (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l' p)"
      proof (intro allI impI)
        fix t l p
        assume in1:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st') =
Some (KStoptr p)"
        have inNew:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e')"
          using denLink in1 by (meson fmlookup_ran_iff)
        from fiFinal inNew in1 obtain t' l' where newPtr:
            "(type.Storage t', Storeloc l') |∈| fmran (Denvalue e')"
            and comp:"CompStoType t' t l' p"
          unfolding fullyInitialised_def by blast
        from newPtr obtain iid where iidDef:"Denvalue e' $$ iid = Some (type.Storage t', Storeloc
l')"
          using fmlookup_ran_iff by fast
        have oldLookup:"Denvalue e_v $$ iid = Some (type.Storage t', Storeloc l')"
        proof (cases "iid = id0")
          case True
          have "∀t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
            using decl_env_storlocs_unchanged[OF 1(2)] by simp
          then show ?thesis using iidDef True by blast
        next
          case False
          then show ?thesis using decl_env_not_i[OF 1(2) iidDef] by blast
        qed
        have oldRan:"(type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v)"
```

```
        using oldLookup by (meson fmranI)
      show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l' p"
        using oldRan comp by blast
    qed
    show "fullyInitialised e_v (Accounts st') (Stack st')"
      unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
  next
    fix locs t
    assume in1:"accessTypeStore locs (Memory st) = Some t"
    then have same:"accessTypeStore locs (Memory st) = accessTypeStore locs mem'"
      using tsDecl in1 by simp
    then have "accessTypeStore locs (mem') = Some t"
      using in1 by simp
    then show "accessTypeStore locs (Memory st') = Some t"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix locs v
    assume in1:"accessStore locs (Memory st) = Some (MPointer v)"
    then have same:"accessStore locs (Memory st) = accessStore locs mem'"
      using tsDecl by simp
    then have "accessStore locs (mem') = Some (MPointer v)"
      using in1 by simp
    then show "∃v'. accessStore locs (Memory st') = Some (MPointer v')"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix locs v
    assume in1:"accessStore locs (Memory st) = Some (MValue v)"
    then have same:"accessStore locs (Memory st) = accessStore locs mem'"
      using tsDecl unfolding ncpOMemInDMem_def by simp
    then have "accessStore locs (mem') = Some (MValue v)"
      using in1 by simp
    then show "∃v'. accessStore locs (Memory st') = Some (MValue v')"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix i loc
    assume in1:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) "
      and in2:" accessStore loc (Memory st) = None"
    moreover have "Toploc (Memory st) ≤ Toploc mem'" using tsDecl unfolding ncpOMemInDMem_def by
simp
    then have same:"accessStore loc (Memory st) = accessStore loc mem'"
      using tsDecl in1 by fastforce
    moreover have "i < Toploc (mem') ∧ LSubPrefL2 loc (ShowL_nat i)"
      using calculation in1
      using ‹Toploc (Memory st) ≤ Toploc mem'› order.strict_trans2 by blast
    ultimately have "accessStore loc mem' = None ⟶
      accessStore loc (Memory st') = None" using IH1 unfolding StateInvariant_def by simp
    then show "accessStore loc (Memory st') = None" using in2 same by simp

  next
    show "Toploc (Memory st) ≤ Toploc (Memory st')"
      using 1 IH1 unfolding StateInvariant_def using tsDecl by simp
  qed
 qed
next
 case (10 id0 tp ex' s e_v cd st)
 show ?case
 proof(cases rule:blockSome[OF 10(3)])
  case (1 v t g cd' mem' sck' e')
  have a10:"assert Gas (λst. costs (BLOCK (id0, tp, Some ex') s) e_v cd st < state.Gas st) st = Normal
((), st)" using 1 by simp
  obtain sgas where sgasDef:"sgas = state.Gas st - costs (BLOCK (id0, tp, Some ex') s) e_v cd st" by
simp
  then have a20:"modify (λst. st⦇Gas := state.Gas st - costs (BLOCK (id0, tp, Some ex') s) e_v cd
st⦈) st = Normal ((), (st⦇Gas := sgas⦈ ))" by simp
```

```
    have a25:"toState (local.expr ex' e_v cd) (st⦇Gas := sgas⦈ ) = Normal ((v, t), (st⦇Gas := g⦈))"
using 1 sgasDef by auto
    have a30:"option Err (λst. decl id0 tp (Some (v, t)) False cd (Memory st) (Storage st (Address e_v))
(cd, Memory st, Stack st, e_v)) (st⦇Gas := g⦈ )
           = Normal ((cd', mem', sck', e'), (st⦇Gas := g⦈ )) " using 1 by simp
    have a40:"modify (λst. st⦇Stack := sck', Memory := mem'⦈) (st⦇Gas := g⦈) = Normal ((), st⦇Gas :=
g, Stack := sck', Memory := mem'⦈)"
      by simp

    have "∃c ct dud.
       Type (Accounts st (Address e_v)) = Some (atype.Contract c) ∧
       environment.Contract e_v = c ∧
       ep $$ c = Some (ct, dud) ∧
       (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc id))) ∧
       (∀id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
       (∀t l p.
           (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
           (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l' p))"
     using sgasDef 10(4) unfolding fullyInitialised_def by simp
    moreover have "fullyInitialised e_v (Accounts (st⦇Gas := sgas⦈)) (Stack (st⦇Gas := sgas⦈))"
     by (simp add: "10.prems"(3))
    moreover have tsSgas:"TypeSafe e_v (Accounts (st⦇Gas := sgas⦈)) (Stack (st⦇Gas := sgas⦈)) (Memory
       (st⦇Gas := sgas⦈)) (Storage (st⦇Gas := sgas⦈)) cd" using 10(2) by simp
    ultimately have tcTV:"exprResultCon e_v cd (st⦇Gas := sgas⦈) v t"
     using 1(2) exprTypeconInduct(3)[of ex' "e_v" cd "st⦇Gas := sgas⦈" sgas v t g] sgasDef by blast
    then have tcTV':"valueCon e_v cd (st⦇Gas := g⦈) v t"
     by (cases t, simp+)

    have tsDecl:"TypeSafe e' (Accounts (st⦇Gas := g⦈)) sck' mem' (Storage (st⦇Gas := g⦈)) cd' ∧
    (¬ False ⟶
     (∀locs tp. MCon tp (Memory (st⦇Gas := g⦈)) locs ⟶ MCon tp mem' locs) ∧
     ncpDenvalueLimit e' e_v sck' (Stack (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈)) ∧
     ncpOMemInDMem (Memory (st⦇Gas := g⦈)) mem' ∧ ncpElementsNoSubPref (Memory (st⦇Gas := g⦈)) mem'
∧ ncpNewSelfPoint (Memory (st⦇Gas := g⦈)) mem') ∧ Toploc (Memory (st⦇Gas := g⦈)) ≤ Toploc mem'"
    proof -
     have tsgas:"TypeSafe e_v (Accounts (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))
(Storage (st⦇Gas := g⦈)) cd"
       using 10(2) by simp
      have "ncpDenvalueLimit e_v e_v (Stack (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈)) (Memory (st⦇Gas :=
g⦈))"
       unfolding ncpDenvalueLimit_def by auto
      moreover have "ncpOMemInDMem (Memory (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))" unfolding
ncpOMemInDMem_def by auto
      moreover have "ncpElementsNoSubPref (Memory (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))"
       using ncpElementsNoSubPref_def ncpElementsNoSubPref_sameMem tsgas by blast
      moreover have "ncpNewSelfPoint (Memory (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))"
       unfolding ncpNewSelfPoint_def by auto
      ultimately have acc0:"¬ False ⟶
    (∀locs tp. MCon tp (Memory (st⦇Gas := g⦈)) locs ⟶ MCon tp (Memory (st⦇Gas := g⦈)) locs) ∧
    Toploc (Memory (st⦇Gas := g⦈)) ≤ Toploc (Memory (st⦇Gas := g⦈)) ∧
    ncpDenvalueLimit e_v e_v (Stack (st⦇Gas := g⦈)) (Stack (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈)) ∧
    ncpOMemInDMem (Memory (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈)) ∧
    ncpElementsNoSubPref (Memory (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈)) ∧ ncpNewSelfPoint (Memory
       (st⦇Gas := g⦈)) (Memory (st⦇Gas := g⦈))" by blast
      have acc1:"∀struct.
     t = type.Memory struct ⟶
     (∃stloc tp'' p.
         (type.Memory tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
         accessStore stloc (Stack (st⦇Gas := g⦈)) = Some (KMemptr p) ∧
         (tp'' = struct ∧ v = KMemptr p ∨
          (∃len arr. p ≠ extractValueType v ∧ tp'' = MTArray len arr ∧ CompMemType (Memory (st⦇Gas :=
g⦈)) len arr struct p (extractValueType v))))"
       using tcTV by (cases t, blast+; fastforce)
```

```
      have acc2:"∀struct.
     t = Calldata struct ⟶
     (∃stloc tp'' p.
         (Calldata tp'', Stackloc stloc) |∈| fmran (Denvalue e_v) ∧
         accessStore stloc (Stack (st(|Gas := g|))) = Some (KCDptr p) ∧
         (tp'' = struct ∧ v = KCDptr p ∨ (∃len arr. p ≠ extractValueType v ∧ tp'' = MTArray len arr
∧ CompMemType cd len arr struct p (extractValueType v))))"
        using tcTV by (cases t, blast+; fastforce)
      have acc3:"∀struct.
       tp = type.Storage struct ⟶
       (∀locs tp. SCon tp locs (Storage (st(|Gas := g|)) (Address e_v)) ⟶ SCon tp locs (Storage (st(|Gas
:= g|)) (Address e_v))) ∧
       (∃stloc tp''.
           (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
           (tp'' = struct ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' struct
stloc (extractValueType v)))"
        using tcTV
       proof(cases t)
        case (Value x1)
        then have "∀struct. tp ≠ type.Storage struct " using 1(3) decl_KValue_tp_match
          by blast
        then show ?thesis by simp
       next
        case (Calldata x2)
        then have "∀struct. tp ≠ type.Storage struct " using 1(3) decl_Calldata_tp_match
          by blast
        then show ?thesis by simp
       next
        case (Memory x3)
        then have "∀struct. tp ≠ type.Storage struct " using 1(3) decl_Memory_tp_match
          by blast
        then show ?thesis by simp
       next
        case (Storage x4)
        then have sc:"SCon x4 (extractValueType v) (Storage (st(|Gas := sgas|)) (Address e_v)) ∧
              (∃xx. v = KStoptr xx) ∧
              (∃stloc tp''.
                  (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
                  tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4
stloc (extractValueType v))"
          using tcTV by auto
         have tt:"(∃x t' t''. x4 = STArray x t' ∧ cps2mTypeCompatible (STArray x t') (MTArray x t'')∧
tp = type.Memory (MTArray x t'')) ∨ tp = type.Storage x4"
          using decl_Memory_tp_options[] 1(3) Storage by simp
        then show ?thesis
        proof(cases "tp = type.Storage x4")
          case True
          moreover have "(∀locs tp. SCon tp locs (Storage (st(|Gas := g|)) (Address e_v)) ⟶ SCon
              tp locs (Storage (st(|Gas := g|)) (Address e_v)))" by auto
          moreover have "(Stack (st(|Gas := sgas|))) = (Stack (st(|Gas := g|)))" by auto
          then have "(∃stloc tp''.
           (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
           tp'' = x4 ∧ v = KStoptr stloc ∨ extractValueType v ≠ stloc ∧ CompStoType tp'' x4 stloc
(extractValueType v))"
            using sc True Storage by presburger
          ultimately show ?thesis
            using Storage tcTV by auto
        next
          case False
          then have "(∃x t' t''. x4 = STArray x t' ∧ cps2mTypeCompatible (STArray x t') (MTArray x
t'')∧ tp = type.Memory (MTArray x t''))"
            using tt by simp
          then show ?thesis by auto
        qed
```

```
      qed
      have " TypeSafe e' (Accounts (st(|Gas := g|))) sck' mem' (Storage (st(|Gas := g|))) cd' ∧
    (¬ False ⟶
     (∀ locs tp. MCon tp (Memory (st(|Gas := g|))) locs ⟶ MCon tp mem' locs) ∧
     Toploc (Memory (st(|Gas := g|))) ≤ Toploc mem' ∧
     ncpDenvalueLimit e' e_v sck' (Stack (st(|Gas := g|))) (Memory (st(|Gas := g|))) ∧
     ncpOMemInDMem (Memory (st(|Gas := g|))) mem' ∧
     ncpElementsNoSubPref (Memory (st(|Gas := g|))) mem' ∧
     ncpNewSelfPoint (Memory (st(|Gas := g|))) mem')
    "
       using typeSafeDecl[OF tsgas tcTV' tsgas 1(3) acc0 acc1 acc2 ] acc3 by fastforce
      then show ?thesis by simp
    qed


    then have a50:"TypeSafe e' (Accounts (st(|Gas := g, Stack := sck', Memory := mem'|))) (Stack (st(|Gas
:= g, Stack := sck', Memory := mem'|)))
   (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) (Storage (st(|Gas := g, Stack := sck', Memory
:= mem'|))) cd'"
      using tsDecl by simp

    have fiDec:" fullyInitialised e' (Accounts (st(|Gas := g, Stack := sck', Memory := mem'|))) (Stack
(st(|Gas := g, Stack := sck', Memory := mem'|)))"
    proof -
      have sameStuff:"Address e' = Address e_v ∧ Contract e' = Contract e_v" using 1(3) decl_env by simp
      have fiOld:"
        ∃ c ct dud.
          Type (Accounts st (Address e_v)) = Some (atype.Contract c) ∧
          Contract e_v = c ∧
          ep $$ c = Some (ct, dud) ∧
          (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)))
∧
          (∀ id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
          (∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p))"
        using 10(4) unfolding fullyInitialised_def by blast
      then obtain c_fi ct_fi dud_fi where fiC:"
          Type (Accounts st (Address e_v)) = Some (atype.Contract c_fi) ∧
          Contract e_v = c_fi ∧
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀ id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
        and fiPtr:"∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p)"
        by blast

      have cfi:"Type (Accounts (st(|Gas := g, Stack := sck', Memory := mem'|)) (Address e')) = Some
(atype.Contract c_fi) ∧
          Contract e' = c_fi ∧ ep $$ c_fi = Some (ct_fi, dud_fi)"
        using fiC sameStuff by simp

      have mapfi:"∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e' $$ id = Some (type.Storage v,
Storeloc id))"
      proof (intro allI)
        fix id v
        show "(ct_fi $$ id = Some (Var v)) = (Denvalue e' $$ id = Some (type.Storage v, Storeloc id))"
        proof
```

```
          assume hct:"ct_fi $$ id = Some (Var v)"
          then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)"
            using fiC by blast
          then show "Denvalue e' $$ id = Some (type.Storage v, Storeloc id)"
            using decl_env_monotonic[OF 1(3)] by blast
        next
          assume hnew:"Denvalue e' $$ id = Some (type.Storage v, Storeloc id)"
          show "ct_fi $$ id = Some (Var v)"
          proof (cases "id = id0")
            case True
            have "∀t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
              using decl_env_storlocs_unchanged[OF 1(3)] by simp
            then have "Denvalue e_v $$ id0 = Some (type.Storage v, Storeloc id0)"
              using hnew True by blast
            then show ?thesis using fiC True by blast
          next
            case False
            then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)"
              using decl_env_not_i[OF 1(3)] hnew by blast
            then show ?thesis using fiC by blast
          qed
        qed
      qed

      have locfi:"∀id v loc. Denvalue e' $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
      proof (intro allI impI)
        fix id v loc
        assume hnew:"Denvalue e' $$ id = Some (type.Storage v, Storeloc loc)"
        show "id = loc"
        proof (cases "id = id0")
          case True
          have "∀t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
            using decl_env_storlocs_unchanged[OF 1(3)] by simp
          then have "Denvalue e_v $$ id0 = Some (type.Storage v, Storeloc loc)"
            using hnew True by blast
          then have "id0 = loc" using fiC by blast
          then show ?thesis using True by simp
        next
          case False
          then have "Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc)"
            using decl_env_not_i[OF 1(3)] hnew by blast
          then show ?thesis using fiC by blast
        qed
      qed

      have ptrfi:"∀tt l p.
          (type.Storage tt, Stackloc l) |∈| fmran (Denvalue e') ∧
          accessStore l (Stack (st⦇Gas := g, Stack := sck', Memory := mem'⦈)) = Some (KStoptr p) ⟶
          (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e') ∧ CompStoType t' tt l' p)"
      proof (intro allI impI)
        fix tt l p
        assume in1:"(type.Storage tt, Stackloc l) |∈| fmran (Denvalue e')∧accessStore l (Stack (st⦇Gas
:= g, Stack := sck', Memory := mem'⦈)) = Some (KStoptr p)"
        then obtain idd where iddDef: "Denvalue e' $$ idd = Some (type.Storage tt, Stackloc l)" by
auto
        show " ∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e') ∧ CompStoType t' tt l'
p"
        proof(cases "idd = id0")
          case True
          then have ddd:"Denvalue e' $$ id0 = Some (type.Storage tt, Stackloc l)" using iddDef by simp
          then show ?thesis
          proof(cases "Denvalue e_v $$ id0")
```

```
            case None
            have ldef:"l = (ShowL_nat (Toploc (Stack (st(|Gas := g|)))))" using decl_stack_top[OF 1(3)
None] iddDef True by simp
            then obtain p2 where p2Def:"Some (v, t) = Some (KStoptr p2, type.Storage tt)"
              using ddd True decl_storage_tp[OF 1(3) None, of tt ] by blast
            then have dcc:"decl id0 tp (Some (KStoptr p2, type.Storage tt)) False cd (Memory (st(|Gas :=
g|))) (Storage (st(|Gas := g|)) (Address e_v))
       (cd, Memory (st(|Gas := g|)), Stack (st(|Gas := g|)), e_v) =
      Some (cd', mem', sck', e')" using 1(3) True by simp
            then have "v = KStoptr p" using decl_stack_topLoc[OF dcc None] using in1 ldef p2Def by
auto
            then have "(∃ stloc tp''.
            (type.Storage tp'', Storeloc stloc) |∈| fmran (Denvalue e_v) ∧
            (tp'' = tt ∧ v = KStoptr stloc ∨ p ≠ stloc ∧ CompStoType tp'' tt stloc p))"
              using tcTV 1(3) p2Def by auto
            then show ?thesis using decl_env_monotonic[OF 1(3)]
              by (metis CompStoType_sameLocNdTyp stackvalue.inject(4) ‹v = KStoptr p› fmlookup_ran_iff)
          next
            case (Some a)
            then have "e_v = e' ∧ sck' = Stack st" using decl_some_same[OF 1(3)] by auto
            then have oldIn:"(type.Storage tt, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l
(Stack st) = Some (KStoptr p)"
              using in1 by simp
            then show ?thesis using fiPtr
              using ‹e_v = e' ∧ sck' = Stack st› by blast
          qed
        next
          case False
          then have oldD:"Denvalue e_v $$ idd = Some (type.Storage tt, Stackloc l)"
            using iddDef in1 1 decl_env_not_i[OF 1(3)] by blast
          then show ?thesis
          proof(cases "l = ShowL_nat (Toploc (Stack st))")
            case True
            then show ?thesis
              by (metis "10.prems"(1) True oldD snd_conv stackLocs_imp_NotDen TypeSafe_def)
          next
            case False
            then have "accessStore l (Stack (st(|Gas := g, Stack := sck', Memory := mem'|))) =
accessStore l (Stack (st))"
              using decl_stack_change[OF 1(3)] by simp
            then have oldAcc:"accessStore l (Stack st) = Some (KStoptr p)" using in1 by simp
            from fiPtr oldD oldAcc obtain t' l' where t'Def: "(type.Storage t', Storeloc l') |∈|
fmran (Denvalue e_v) ∧ CompStoType t' tt l' p"
              by (meson fmranI)
            then have "(type.Storage t', Storeloc l') |∈| fmran (Denvalue e')"
              using decl_env_monotonic[OF 1(3)] by (meson fmlookup_ran_iff)
            then show ?thesis using t'Def by auto
          qed
        qed
      qed
      show ?thesis
        unfolding fullyInitialised_def using cfi mapfi locfi ptrfi by blast
    qed
    have sameCD:"cd = cd'" using decl_env_false_same_cd[OF 1(3)] by simp
    have IH1:" StateInvariant e' (st(|Gas := g, Stack := sck', Memory := mem'|)) st' cd'"
      using 10(1)[OF a10 a20 a25 _ a30, of cd' "(mem', sck', e')" mem' "(sck', e')" sck' e', OF _ _ _ _
a40 a50 _ fiDec, of st'] 1(4) sgasDef by simp

    have denLink:"∀ ii x. Denvalue e_v $$ ii = Some x ⟶ Denvalue e' $$ ii = Some x" using
decl_env_monotonic[OF 1(3)] by auto
    have sameAdd:"Address e' = Address e_v"using 1(3) decl_env by simp
    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e_v (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
```

```
TypeSafe_def
      proof intros
        show "AddressTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        show "safeContract (Accounts st') (Storage st')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast
      next
        show "unique_locations (Denvalue e_v)" using 10(2) unfolding TypeSafe_def by simp
      next
        have cpNew:"(∀ tp1 tp2 l1 l2 l1' l2' stl1 stl2.
       (type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e') ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2) ⟶
       (if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2 else if TypedStoSubpref stl1
stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True))"
          using IH1 unfolding StateInvariant_def TypeSafe_def compPointers_def by auto
        show "compPointers (Stack st') (Denvalue e_v)" unfolding compPointers_def
        proof intros
          fix tp1 tp2 l1 l2 l1' l2' stl1 stl2
          assume a0:"(type.Storage tp1, l1) |∈| fmran (Denvalue e_v) ∧
       (type.Storage tp2, l2) |∈| fmran (Denvalue e_v) ∧
       (l1 = Stackloc l1' ∧ accessStore l1' (Stack st') = Some (KStoptr stl1) ∨ l1 = Storeloc stl1) ∧
       (l2 = Stackloc l2' ∧ accessStore l2' (Stack st') = Some (KStoptr stl2) ∨ l2 = Storeloc stl2)"
          then obtain i1 i2 where iDefs:" (Denvalue e_v) $$ i1 = Some (type.Storage tp1, l1)
                                  ∧ (Denvalue e_v) $$ i2 = Some (type.Storage tp2, l2)" by blast
          then have "(type.Storage tp1, l1) |∈| fmran (Denvalue e') ∧ (type.Storage tp2, l2) |∈| fmran
(Denvalue e')" using a0
            by (metis denLink fmlookup_ran_iff)
          then show "if TypedStoSubpref stl2 stl1 tp1 then CompStoType tp1 tp2 stl1 stl2
               else if TypedStoSubpref stl1 stl2 tp2 then CompStoType tp2 tp1 stl2 stl1 else True"
            using cpNew using a0 by blast
        qed
      next
        show "balanceTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        have "envAddressesWellFormed e_v" using 10(2) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e_v)" and "addressFormat (Sender e_v)" by simp+
      next
        show "svalueTypes (Svalue e_v)" using 10(2) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        show "lessThanTopLocs cd" using 10(2) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
      next
        have oldTC:"typeCompat (Denvalue e') (Stack st') (Memory st') (Storage st' (Address e')) cd'"
          using IH1 unfolding StateInvariant_def TypeSafe_def by blast
        show "typeCompat (Denvalue e_v) (Stack st') (Memory st') (Storage st' (Address e_v)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume inDen:"(t, l) |∈| fmran (Denvalue e_v)"
          then have inDenN:"(t, l) |∈| fmran (Denvalue e')"
            using denLink by (meson fmlookup_ran_iff)

          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e_v)) cd t l"
          proof(cases l)
            case (Stackloc x1)
```

```
            then have oldTC':"(
             case accessStore x1 (Stack st') of None ⇒ False
              | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
              | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd' stloc | _ ⇒ False)
              | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory st') stloc
| _ ⇒ False)
              | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc (Storage st'
(Address e')) | _ ⇒ False))"
              using oldTC unfolding typeCompat_def using inDenN by fastforce
            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Stackloc by
fastforce
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1')
                then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Some
Stackloc by force
              next
                case (KCDptr x2)
                then obtain struct where structDef:"t = Calldata struct ∧ MCon struct cd' x2"
                  using oldTC' unfolding typeCompat_def using inDenN sameAdd Some Stackloc by (cases
t; auto)
                then show ?thesis using tsDecl KCDptr Some Stackloc sameCD by simp
              next
                case (KMemptr x3)
                then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd Some
Stackloc by fastforce
              next
                case (KStoptr x4)
                then have "case t of type.Storage struct ⇒ SCon struct x4 (Storage st' (Address e')) |
_ ⇒ False"
                  using oldTC' using inDenN sameAdd Some Stackloc by (auto split:option.splits
type.splits stackvalue.splits denvalue.splits)
                then obtain struct where "t = type.Storage struct ∧ SCon struct x4 (Storage st'
(Address e'))" by (cases t; auto)
                then show ?thesis using Some Stackloc KStoptr
                  by (simp add: sameAdd)
              qed
            qed
          next
            case (Storeloc x2)
            then show ?thesis using oldTC unfolding typeCompat_def using inDenN sameAdd
              by (metis (no_types, lifting) denvalue.simps(6) type.exhaust type.simps(17,18,19,20))
          qed
        qed
      next

        show "denvalueTypeCorrectness e_v (Stack st') (Memory st')"
          unfolding denvalueTypeCorrectness_def
        proof intros
          fix t l ptr_loc sub_loc
          assume in1:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st')
= Some (KMemptr ptr_loc)"
          then have in2:"(type.Memory t, Stackloc l) |∈| fmran (Denvalue e')" using denLink
            by (meson fmlookup_ran_iff)
          then show "(case t of MTArray len arr ⇒ ∀i<len. accessTypeStore (hash ptr_loc (ShowL_nat i))
(Memory st') = Some arr
        | MTValue val ⇒ accessTypeStore ptr_loc (Memory st') = Some (MTValue val))"
            using IH1 unfolding StateInvariant_def TypeSafe_def denvalueTypeCorrectness_def
            using in1 in2 by blast
```

```
      qed
    next
      show "subPrefixStructuralConsistency (Memory st')" using IH1 unfolding StateInvariant_def
TypeSafe_def by blast
    next
      show "SomeValSomeTyp (Memory st')" using IH1 unfolding StateInvariant_def TypeSafe_def by
blast
    qed

  next
    have fiOld:"
      ∃ c ct dud.
        Type (Accounts st (Address e_v)) = Some (atype.Contract c) ∧
        Contract e_v = c ∧
        ep $$ c = Some (ct, dud) ∧
        (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc id)))
∧
        (∀ id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
        (∀ t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
            (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p))"
      using 10(4) unfolding fullyInitialised_def by blast
    then obtain c_fi ct_fi dud_fi where fiC:"
        Type (Accounts st (Address e_v)) = Some (atype.Contract c_fi) ∧
        Contract e_v = c_fi ∧
        ep $$ c_fi = Some (ct_fi, dud_fi) ∧
        (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_v $$ id = Some (type.Storage v, Storeloc
id))) ∧
        (∀ id v loc. Denvalue e_v $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
      and fiPtr:"∀ t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
            (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l'
p)"
      by blast
    have typeNew:"Type (Accounts st' (Address e_v)) = Some (atype.Contract c_fi)"
      using fiC atype_same "10.prems"(2,3) statement_with_gas.atype_same statement_with_gas_axioms by
blast
    have fiFinal:"fullyInitialised e' (Accounts st') (Stack st')"
      using IH1 unfolding StateInvariant_def by blast
    have ptrNew:"∀ t l p.
        (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
        (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l' p)"
    proof (intro allI impI)
      fix t l p
      assume in1:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e_v) ∧ accessStore l (Stack st') =
Some (KStoptr p)"
      have inNew:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e')"
        using denLink in1 by (meson fmlookup_ran_iff)
      from fiFinal inNew in1 obtain t' l' where newPtr:
          "(type.Storage t', Storeloc l') |∈| fmran (Denvalue e')"
        and comp:"CompStoType t' t l' p"
        unfolding fullyInitialised_def by blast
      from newPtr obtain iid where iidDef:"Denvalue e' $$ iid = Some (type.Storage t', Storeloc
l')"
        using fmlookup_ran_iff by fast
      have oldLookup:"Denvalue e_v $$ iid = Some (type.Storage t', Storeloc l')"
      proof (cases "iid = id0")
        case True
        have "∀ t l. Denvalue e' $$ id0 = Some (t, Storeloc l) ⟶ Denvalue e_v $$ id0 = Some (t,
Storeloc l)"
```

```
          using decl_env_storlocs_unchanged[OF 1(3)] by simp
        then show ?thesis using iidDef True by blast
      next
        case False
        then show ?thesis using decl_env_not_i[OF 1(3) iidDef] by blast
      qed
      have oldRan:"(type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v)"
        using oldLookup by (meson fmranI)
      show "∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_v) ∧ CompStoType t' t l' p"
        using oldRan comp by blast
    qed
    show "fullyInitialised e_v (Accounts st') (Stack st')"
      unfolding fullyInitialised_def using fiC fiPtr ptrNew typeNew by blast
  next
    fix locs t
    assume in1:"accessTypeStore locs (Memory st) = Some t"
    then have "∃v. accessStore locs (Memory st) = Some v"
      using 10(2) unfolding TypeSafe_def SomeValSomeTyp_def by blast
    then obtain i where idef:"i < Toploc (Memory st) ∧ LSubPrefL2 locs (ShowL_nat i) "
      using 10(2) unfolding TypeSafe_def lessThanTopLocs_def by blast
    then have same:"accessTypeStore locs (Memory st) = accessTypeStore locs mem'"
      using tsDecl unfolding ncpOMemInDMem_def by simp
    then have "accessTypeStore locs (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) = Some t"
      using in1 by simp
    then show "accessTypeStore locs (Memory st') = Some t"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix locs v
    assume in1:"accessStore locs (Memory st) = Some (MPointer v)"
    then obtain i where idef:"i < Toploc (Memory st) ∧ LSubPrefL2 locs (ShowL_nat i) "
      using 10(2) unfolding TypeSafe_def lessThanTopLocs_def by blast
    then have same:"accessStore locs (Memory st) = accessStore locs mem'"
      using tsDecl unfolding ncpOMemInDMem_def by simp
    then have "accessStore locs (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) = Some
(MPointer v)"
      using in1 by simp
    then show "∃v'. accessStore locs (Memory st') = Some (MPointer v')"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix locs v
    assume in1:"accessStore locs (Memory st) = Some (MValue v)"
    then obtain i where idef:"i < Toploc (Memory st) ∧ LSubPrefL2 locs (ShowL_nat i) "
      using 10(2) unfolding TypeSafe_def lessThanTopLocs_def by blast
    then have same:"accessStore locs (Memory st) = accessStore locs mem'"
      using tsDecl unfolding ncpOMemInDMem_def by simp
    then have "accessStore locs (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) = Some (MValue
v)"
      using in1 by simp
    then show "∃v'. accessStore locs (Memory st') = Some (MValue v')"
      using in1 IH1 unfolding StateInvariant_def by simp
  next
    fix i loc
    assume in1:"i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) "
      and in2:" accessStore loc (Memory st) = None"
    then have same:"accessStore loc (Memory st) = accessStore loc mem'"
      using tsDecl unfolding ncpOMemInDMem_def by simp
    moreover have "Toploc (Memory st) ≤ Toploc mem'" using tsDecl unfolding ncpOMemInDMem_def by
simp
    moreover have "i < Toploc (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) ∧ LSubPrefL2
loc (ShowL_nat i)"
      using calculation in1 by auto
    ultimately have "accessStore loc (Memory (st(|Gas := g, Stack := sck', Memory := mem'|))) = None
⟶
      accessStore loc (Memory st') = None" using IH1 unfolding StateInvariant_def by simp
```

```
      then show "accessStore loc (Memory st') = None" using in2 same by simp

    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')"
        using 1 IH1 unfolding StateInvariant_def using tsDecl by simp
    qed
  qed
next
  case (11 i xe val e cd st)
  show ?case
  proof(cases rule:new[OF 11(3)])
    case (1 v t g ct cn fb e_l cd_l k_l m_l g' acc st'' v')
    then have a10:"assert Gas (λst. costs (NEW i xe val) e cd st < state.Gas st) st = Normal ((), st)"
by simp
    let ?sgas = "st(|Gas := state.Gas st - costs (NEW i xe val) e cd st|)"
    have a20:"modify (λst. st(|Gas := state.Gas st - costs (NEW i xe val) e cd st|)) st = Normal ((),
?sgas)" by simp
    let ?newAdd = "hash_version (Address e) (ShowL_nat (Contracts (Accounts ?sgas (Address e))))"
    have notSame:"(Address e) ≠ ?newAdd"
      by (metis hash_version_inequality)
    have formatAdd:"addressFormat (hash_version (Address e) (ShowL_nat (Contracts (Accounts (st(|Gas :=
state.Gas st - costs (NEW i xe val) e cd st|)) (Address e)))))"
      unfolding addressFormat_def typeCon.simps
    proof -
      have "CHR ''.'' ∉ set (literal.explode (Address e))"
        using 11(2) unfolding TypeSafe_def addressFormat_def typeCon.simps checkAddress_def by blast
      then have "CHR ''.'' ∉ set (literal.explode (STR ''-'' + Address e))"
        by (simp add: Literal.rep_eq add_Literal_assoc)
      moreover have "CHR ''.'' ∉ set (literal.explode (ShowL_nat (Contracts (Accounts (st(|Gas :=
state.Gas st - costs (NEW i xe val) e cd st|)) (Address e)))))"
        using ShowLNatDot by blast

      ultimately have "CHR ''.'' ∉ set
                       (literal.explode ((ShowL_nat (Contracts (Accounts (st(|Gas := state.Gas st - costs
(NEW i xe val) e cd st|)) (Address e))))
                       + (STR ''-'' + Address e)))"
        by (simp add: plus_literal.rep_eq)
      then show "checkAddress (hash_version (Address e) (ShowL_nat (Contracts (Accounts (st(|Gas :=
state.Gas st - costs (NEW i xe val) e cd st|)) (Address e)))))"
        unfolding checkAddress_def hash_version_def by blast
    qed
    let ?updState = "st(|Gas := g, Accounts := (Accounts st)((?newAdd) := (|Bal = ShowL_int 0, Type = Some
(atype.Contract i), Contracts = 0|)),
                                              Storage := (Storage st)((?newAdd) := {$$})|)"
    obtain uus where uusDef:"uus = ?updState" by blast
    have sameUpd:"Memory ?updState = Memory st ∧ Stack ?updState = Stack st" by auto
    have a30:"applyf (λst. hash_version (Address e) (ShowL_nat (Contracts (Accounts st (Address e)))))
(?sgas)
                  = Normal (hash_version (Address e) (ShowL_nat (Contracts (Accounts ?sgas (Address
e)))) , (?sgas))" using 1 by auto
    have a40:" assert Err (λst. Type (Accounts st ?newAdd) = None) (?sgas)
                        = Normal ((), ?sgas)" using 1 by simp
    have a50:"toState (local.expr val e cd) (?sgas) = Normal ((KValue v, Value t), st(|Gas := g|))" us-
ing 1 by auto
    have a60:"(case (KValue v, Value t) of (KValue v, Value t) ⇒ return (v, t) | (KValue v, _) ⇒
throw Err
                | (_, b) ⇒ throw Err) (st(|Gas := g|)) = Normal ((v,t), st(|Gas := g|))" using 1 by
simp
    have a65:"option Err (λ_. convert t (TUInt b256) v) (st(|Gas := g|)) = Normal (v', st(|Gas := g|))"
using 1 by simp
    have a70:"option Err (λ_. ep $$ i) (st(|Gas := g|)) = Normal ((ct, cn, fb), st(|Gas := g|))" using 1
by simp
    obtain folded where a80: "folded = ffold_init ct (emptyEnv (?newAdd) i (Address e) v') (fmdom ct)"
      using 1 by blast
```

```
    have a90:"toState (local.load True (fst cn) xe folded emptyTypedStore emptyStore emptyTypedStore e
cd) ?updState = Normal ((e_l,
        cd_l, k_l, m_l), ?updState(|Gas := g'|))"
      using 1 a80 by simp


    have a110:" option Err (λst. transfer (Address e) ?newAdd v' (Accounts st)) (?updState(|Gas:=g'|)) =
Normal (acc, ?updState(|Gas:=g'|))"
      using 1 by auto
    have a120:"applyf (λst. (Stack st, Memory st)) (?updState(|Gas:=g'|)) = Normal ((Stack st, Memory
st), ?updState(|Gas:=g'|))"
      by simp
    have a130:"modify (λst'. st'(|Accounts := acc, Stack := k_l, Memory := m_l|)) (?updState (|Gas:=g'|))
           = Normal ((), ?updState(|Gas:=g', Accounts := acc, Stack := k_l, Memory := m_l|))"
      by simp

    have BtUus:"balanceTypes (Accounts uus)"
      unfolding balanceTypes_def
    proof intros
      fix adv
      show "typeCon (TUInt b256) (Bal (Accounts uus adv)) "
      proof(cases "adv = ?newAdd")
        case True
        then have " (Bal (Accounts uus adv)) = (ShowL_int 0)" using uusDef by simp
        then show ?thesis unfolding typeCon.simps checkUInt_def
          using Read_ShowL_id by presburger
      next
        case False
        then have "(Bal (Accounts uus adv)) = (Bal (Accounts st adv))" using uusDef by simp
        then show ?thesis using 11(2) unfolding TypeSafe_def balanceTypes_def by simp
      qed
    qed

    have svv':"svalueTypes v'" unfolding svalueTypes_def
      using 1(4,3) 11(2) 11(4)
      using exprTypeconInduct(3)[of ]
        typeSafeConvert[of t v "(TUInt b256)"] convertSame by fastforce

    have btACC:"balanceTypes acc" unfolding balanceTypes_def
    proof -
      have old:"balanceTypes (Accounts (uus))" using BtUus by simp
      have accExp:"(case subBalance (Address e) v' (Accounts (uus)) of None ⇒ None | Some x ⇒
addBalance ?newAdd v' x) = Some acc"
        using 1(7) uusDef unfolding transfer_def by simp
      have v'TC: "typeCon (TUInt b256) v'"
        using svv' unfolding svalueTypes_def by blast

      show "∀adv'. typeCon (TUInt b256) (Bal (acc adv')) "
      proof(cases "?newAdd = (Address e)")
        case same:True
        then show ?thesis using notSame by simp
      next
        case notSame:False
        show ?thesis
        proof
          fix adv'
          show "typeCon (TUInt b256) (Bal (acc adv')) "
          proof(cases "adv' = (Address e)")
            case True

            then have tco:"typeCon (TUInt b256) (Bal ((Accounts uus) adv'))"
              using True old accExp balanceTypes_def by blast
            have "Bal (acc (Address e)) = ShowL_int (ReadL_int (Bal (Accounts (uus) (Address e))) -
(ReadL_int v'))"
```

```
            using transfer_subRead[OF _ _ v'TC ] 1(7) uusDef True notSame tco by auto
          then show ?thesis using True
            using "1"(7,3) Read_ShowL_id tco checkUInt_def transfer_val1 transfer_val2
            by (metis transfer_Bal_typeCon[unfolded transfer_def, OF accExp] old[unfolded
balanceTypes_def])

        next
          case False
          then show ?thesis
          proof(cases "adv' = ?newAdd")
            case True
            then have tco:"typeCon (TUInt b256) (Bal ((Accounts uus) ?newAdd))"
              using True old accExp balanceTypes_def by simp
            then have "Bal (acc ?newAdd) = (ShowL_int ((ReadL_int (Bal (Accounts (uus) ?newAdd))) +
(ReadL_int v')))"
              using transfer_addRead[OF 1(7)_ v'TC ] 1(3) notSame uusDef by auto
            moreover have "typeCon (TUInt b256) (ShowL_int ((ReadL_int (Bal (Accounts (uus) ?newAdd)))
+ (ReadL_int v')))"
              using "1"(7,3) Read_ShowL_id tco checkUInt_def transfer_val1 transfer_val2
transfer_val3
              using transfer_Bal_typeCon[unfolded transfer_def, OF accExp] old calculation unfolding
balanceTypes_def by metis
            ultimately show ?thesis using True by simp
          next
            case f2:False
            show ?thesis using transfer_eq[OF 1(7) False f2] old
              using balanceTypes_def uusDef by simp
          qed
        qed
      qed
    qed

    obtain ii :: environment where iiDef: "i = Contract ii" using 1
      by (metis Environment.select_convs(2))
    have epA:"ep $$ Contract ii = Some (ct, cn, fb)" using 1 iiDef by blast
    have atUus:"AddressTypes (Accounts uus)"
      unfolding AddressTypes_def
    proof intros
      fix adv
      show "case Type (Accounts uus adv) of None ⇒ True | Some EOA ⇒ True | Some (atype.Contract c)
⇒ addressFormat adv ∧ c |∈| fmdom ep "
      proof(cases "Type (Accounts uus adv)")
        case None
        then show ?thesis by simp
      next
        case (Some a)
        then show ?thesis
        proof(cases a)
          case EOA
          then show ?thesis using Some by simp
        next
          case (Contract x2)
          then show ?thesis
          proof(cases "adv = ?newAdd")
            case True
            then have "Type (Accounts uus adv) = Some (atype.Contract i)" using uusDef by auto
            then have "i = x2" using Some Contract by simp
            moreover have "ep $$ i = Some (ct, cn, fb)" using epA iiDef by simp
            moreover have "addressFormat adv" using iiDef epA formatAdd True by simp
            ultimately show ?thesis using Some Contract
              by (metis Option.option.simps(5) atype.simps(5) fmlookup_dom_iff)
          next
            case False
```

```
          then have "(Accounts uus adv) = Accounts st adv" using uusDef by simp
          then show ?thesis using 11(2) unfolding TypeSafe_def AddressTypes_def using Some Contract
            by metis
        qed
      qed
    qed
    have scUus:"safeContract (Accounts uus) (Storage uus)"
      unfolding safeContract_def
    proof intros
      fix e' ct' dud' i' tp
      assume in1:"Type (Accounts uus (Address (e'::environment))) = Some (atype.Contract (Contract e'))
∧
                ep $$ Contract (e'::environment) = Some (ct', dud') ∧
                ct' $$ i' = Some (Var tp)"
      show "SCon tp i' (Storage uus (Address e'))"
      proof(cases "Address e' = ?newAdd")
        case True
        then have "(Storage uus (Address e')) = {$$}" using uusDef by simp
        then show ?thesis
        proof(induction tp arbitrary:i')
          case (STArray x1 tp)
          then show ?case by simp
        next
          case (STMap x1 tp)
          then show ?case by simp
        next
          case (STValue x)
          have "typeCon x (ival x)" by (simp add: ivalTypeCon)
          then show ?case unfolding SCon.simps accessStorage_def using STValue by simp
        qed
      next
        case False
        then have "Storage uus (Address e') = Storage st (Address e')"
          using uusDef by simp
        moreover have "Type (Accounts st (Address e')) = Some (atype.Contract (Contract e'))"
          using False in1 uusDef by simp
        moreover have "ep $$ Contract (e'::environment) = Some (ct', dud')"
          using in1 by simp
        moreover have "ct' $$ i' = Some (Var tp)"
          using in1 by simp
        moreover have "safeContract (Accounts st) (Storage st)"
          using 11(2) unfolding TypeSafe_def by simp
        ultimately show ?thesis
          unfolding safeContract_def
          by presburger
      qed
    qed
    have cohUus:"Type (Accounts uus ?newAdd) = Some (atype.Contract (Contract ii))"
      using uusDef iiDef by simp
    have TSInit:"∀x y. TypeSafe folded (Accounts uus)
                  emptyStore emptyTypedStore (Storage uus) emptyTypedStore ∧
       (Denvalue folded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue folded $$ x = Some y ⟶
           (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    proof -
      have "safeContract (Accounts uus) (Storage uus)" using scUus by simp
      moreover have "balanceTypes (Accounts uus)" using BtUus by simp
      moreover have "svalueTypes v'"
      proof -
        have "typeCon t (extractValueType (KValue v))"
          using 1(4,3) 11(2) 11(4)
          using exprTypeconInduct(3)[of val e cd "(st⦇Gas := state.Gas st - costs (NEW i xe val) e cd
st⦈)" "(state.Gas
              (st⦇Gas := state.Gas st - costs (NEW i xe val) e cd st⦈))" "KValue v" "Value t" g]
```

```
            typeSafeConvert[of t v "(TUInt b256)"] convertSame by simp
        then show ?thesis unfolding svalueTypes_def using 1(3) a65
          by (metis "1"(4) convertSame extractValueType.simps(1) typeSafeConvert)
      qed
      moreover have "lessThanTopLocs emptyStore" using emptyTopLocs by simp
      moreover have "ep $$ Contract ii = Some (ct, cn, fb)" using epA by simp
      moreover have "addressFormat (?newAdd)" using formatAdd by simp
      moreover have "addressFormat (Address e)" using 11(2) unfolding TypeSafe_def by simp
      moreover have "AddressTypes (Accounts uus)" using atUus by simp
      moreover have sp:"subPrefixStructuralConsistency emptyTypedStore" unfolding
subPrefixStructuralConsistency_def accessTypeStore_def emptyTypedStore_def by simp
      moreover have "SomeValSomeTyp emptyTypedStore" unfolding SomeValSomeTyp_def accessTypeStore_def
accessStore_def emptyTypedStore_def by auto
      moreover have "Type (Accounts uus ?newAdd) = Some (atype.Contract (Contract ii))" using cohUus
by simp

      ultimately show "∀x y. TypeSafe folded (Accounts uus)
                emptyStore emptyTypedStore (Storage uus) emptyTypedStore ∧
       (Denvalue folded $$ x = Some y ⟶ snd y = Storeloc x) ∧ (Denvalue folded $$ x = Some y ⟶
           (∃t1. ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
        using a80 unfolding ffold_init_def
        using ffoldInitTypeSafe[of "Accounts uus" "Storage uus" v' emptyTypedStore ii ct "(cn, fb)"
?newAdd "Address e" "fmdom ct"]
        using iiDef typedEmptyTopLocs by blast
    qed
    have a140:"TypeSafe e_l (Accounts (uus⦇Gas:=g', Accounts := acc, Stack := k_l, Memory := m_l⦈))
                            (Stack (uus⦇Gas:=g', Accounts := acc, Stack := k_l, Memory :=m_l⦈))
                            (Memory (uus⦇Gas:=g', Accounts := acc, Stack := k_l, Memory := m_l⦈))
                            (Storage (uus⦇Gas:=g', Accounts := acc, Stack := k_l, Memory := m_l⦈))
                            cd_l ∧ fullyInitialised e_l acc k_l"
    proof -

      have aa1:"load True (fst cn) xe folded emptyTypedStore emptyStore emptyTypedStore e cd ?updState
g =
    Normal ((e_l, cd_l, k_l, m_l), g')" using 1(6) a80 unfolding ffold_init_def by simp
      moreover have tsE:"TypeSafe e (Accounts uus) (Stack uus) (Memory uus) (Storage uus) cd"
        unfolding TypeSafe_def
      proof intros
        show "AddressTypes (Accounts uus)" using atUus by simp
      next
        show "safeContract (Accounts uus) (Storage uus)" using scUus by simp
      next
        show "unique_locations (Denvalue e)" using 11(2) unfolding TypeSafe_def by simp
      next

        show "compPointers (Stack uus) (Denvalue e)"
          using 11(2) unfolding TypeSafe_def
          using sameUpd uusDef by argo
      next
        show "balanceTypes (Accounts uus)" using BtUus by simp
      next
        have "envAddressesWellFormed e" using 11(2) sameUpd uusDef unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using 11(2) sameUpd uusDef unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack uus)" using 11(2) sameUpd uusDef unfolding TypeSafe_def by auto
      next
        show "lessThanTopLocs cd" using 11(2) sameUpd uusDef unfolding TypeSafe_def by auto
      next
        show "lessThanTopLocs (Memory uus)" using 11(2) sameUpd uusDef unfolding TypeSafe_def by auto
      next
        have tcOld:"typeCompat (Denvalue e) (Stack st) (Memory st) (Storage st (Address e)) cd"
          using 11(2) unfolding TypeSafe_def by blast
```

```
        show "typeCompat (Denvalue e) (Stack uus) (Memory uus) (Storage uus (Address e)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume inDen:"(t, l) |∈| fmran (Denvalue e)"

          show "denvalueCon (Stack uus) (Memory uus) (Storage uus (Address e)) cd t l"
          proof(cases l)
            case (Stackloc x1)
            have tcOld':"case accessStore x1 (Stack st) of None ⇒ False
                        | Some (KValue val) ⇒ (case t of Value typ ⇒ typeCon typ val | _ ⇒ False)
                        | Some (KCDptr stloc) ⇒ (case t of Calldata struct ⇒ MCon struct cd stloc | _
⇒ False)
                        | Some (KMemptr stloc) ⇒ (case t of type.Memory struct ⇒ MCon struct (Memory
st) stloc | _ ⇒ False)
                        | Some (KStoptr stloc) ⇒ (case t of type.Storage struct ⇒ SCon struct stloc
(Storage st (Address e)) | _ ⇒ False)"
              using sameUpd Stackloc inDen tcOld unfolding typeCompat_def by force
            then show ?thesis
            proof(cases "accessStore x1 (Stack uus)")
              case None
              then show ?thesis using tcOld' sameUpd uusDef by simp
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1')
                then show ?thesis using tcOld' sameUpd uusDef Some Stackloc by simp
              next
                case (KCDptr x2)

                then show ?thesis using tcOld' sameUpd uusDef Some Stackloc by simp
              next
                case (KMemptr x3)
                then show ?thesis using tcOld' sameUpd uusDef Some Stackloc by (cases t; fastforce)
              next
                case (KStoptr x4)

                then show ?thesis
                proof(cases "Address e = ?newAdd")
                  case True
                  then have empt:"Storage uus (Address e) = {$$}" using uusDef by simp
                  then obtain struct where struct_def:"t = type.Storage struct"
                    using tcOld' sameUpd uusDef Some Stackloc KStoptr by (cases t; fastforce)
                  have "SCon struct x4 (Storage uus (Address e))"
                  proof(induction struct arbitrary:x4)
                    case (STArray x1 tp)
                    then show ?case by simp
                  next
                    case (STMap x1 tp)
                    then show ?case by simp
                  next
                    case (STValue x)
                    have "typeCon x (ival x)" by (simp add: ivalTypeCon)
                    then show ?case unfolding SCon.simps accessStorage_def using STValue empt by simp
                  qed
                  then show ?thesis using tcOld' sameUpd uusDef Some Stackloc KStoptr struct_def by
simp
                next
                  case False
                  then have "Storage uus (Address e) = Storage st (Address e)" using uusDef by simp
                  then show ?thesis using tcOld' sameUpd uusDef Some Stackloc KStoptr by (cases t;
fastforce)
                qed
```

```
            qed
          qed
        next
          case (Storeloc x2)
          then show ?thesis
          proof(cases "Address e = ?newAdd")
            case True
            then have empt:"Storage uus (Address e) = {$$}" using uusDef by simp
            then obtain struct where struct_def:"t = type.Storage struct"
              using sameUpd uusDef Storeloc tcOld inDen unfolding typeCompat_def by (cases t;
fastforce)
            have "SCon struct x2 (Storage uus (Address e))"
            proof(induction struct arbitrary:x2)
              case (STArray x1 tp)
              then show ?case by simp
            next
              case (STMap x1 tp)
              then show ?case by simp
            next
              case (STValue x)
              have "typeCon x (ival x)" by (simp add: ivalTypeCon)
              then show ?case unfolding SCon.simps accessStorage_def using STValue empt by simp
            qed
            then show ?thesis using tcOld inDen unfolding typeCompat_def using sameUpd uusDef
Storeloc struct_def by simp
          next
            case False
            then have "Storage uus (Address e) = Storage st (Address e)" using uusDef by simp
            then show ?thesis using tcOld inDen unfolding typeCompat_def using sameUpd uusDef
Storeloc by (cases t; fastforce)
          qed
        qed
      qed
    next
      show "denvalueTypeCorrectness e (Stack uus) (Memory uus)" using uusDef using 11(2) unfolding
TypeSafe_def by simp
    next
      show "subPrefixStructuralConsistency (Memory uus)" using uusDef using 11(2) unfolding
TypeSafe_def by simp
    next
      show "SomeValSomeTyp (Memory uus)" using uusDef using 11(2) unfolding TypeSafe_def by simp
    qed

    moreover have "TypeSafe folded (Accounts uus) emptyStore emptyTypedStore (Storage uus)
emptyTypedStore" using TSInit
      by blast

    moreover have a6:"fullyInitialised e (Accounts uus) (Stack uus)"
    proof -
      have fiOld:"
        ∃c ct dud.
          Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
          Contract e = c ∧
          ep $$ c = Some (ct, dud) ∧
          (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
          (∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l'
p))"
        using 11(4) unfolding fullyInitialised_def by blast
      then obtain c_fi ct_fi dud_fi where fiC:"
```

```
            Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
            Contract e = c_fi ∧
            ep $$ c_fi = Some (ct_fi, dud_fi) ∧
            (∀id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
            (∀id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
          and fiPtr:"∀t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l'
p)"
          by blast
        have typeNew:"Type (Accounts uus (Address e)) = Some (atype.Contract c_fi)"
          using fiC uusDef notSame by simp
        have ptrNew:"∀t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack uus) = Some
(KStoptr p) ⟶
            (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
          using fiPtr sameUpd uusDef by simp
        show ?thesis
          unfolding fullyInitialised_def using fiC typeNew ptrNew by blast
      qed

      have contractL:"Contract folded = Contract ii" using a80 iiDef by simp

      have ctd:"ep $$ Contract folded = Some (ct, cn, fb)" using a50 contractL 1 a10 iiDef by metis
      then have fi2:"∀id v. (Denvalue folded $$ id = Some (type.Storage v, Storeloc id)) = (ct $$ id =
Some (Var v))"
        using a80 unfolding ffold_init_def using ffoldInit_var_storage_mapping_eq[of ct ?newAdd i
"(Address e)" v' folded] by auto
      moreover have "fullyInitialised e (Accounts ?updState) (Stack ?updState)" using a6 uusDef by
blast
      moreover have fiFolded:"fullyInitialised folded (Accounts uus) emptyStore"
      proof -
        obtain c_fi where cfi:
          "Type (Accounts uus (Address folded)) = Some (atype.Contract c_fi) ∧ Contract folded = c_fi"
          using a80 11(4) uusDef unfolding fullyInitialised_def by simp
        obtain dud_fi where epfi:"ep $$ Contract folded = Some (ct, dud_fi)"
          using ctd by blast
        have mapfi:"∀id v. (ct $$ id = Some (Var v)) = (Denvalue folded $$ id = Some (type.Storage v,
Storeloc id))"
          using fi2 by blast
        have locfi:"∀id v loc. Denvalue folded $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc"
        proof (intro allI impI)
          fix id v loc
          assume h:"Denvalue folded $$ id = Some (type.Storage v, Storeloc loc)"
          have hloc:"Denvalue folded $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage
v, Storeloc loc) = Storeloc id"
            using TSInit by blast
          then show "id = loc" using h by simp
        qed
        have ptrfi:"∀t l p.
            (type.Storage t, Stackloc l) |∈| fmran (Denvalue folded) ∧ accessStore l emptyStore = Some
(KStoptr p) ⟶
            (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue folded) ∧ CompStoType t' t l'
p)"
          by (simp add: accessStore_def emptyStore_def)
        show ?thesis
          unfolding fullyInitialised_def using cfi epfi mapfi locfi ptrfi by blast
      qed

      ultimately have exprTS:" TypeSafe e_l (Accounts ?updState) k_l m_l (Storage ?updState) cd_l ∧
    fullyInitialised e_l (Accounts ?updState) k_l ∧ fullyInitialised e_l
```

```
    (Accounts ?updState) k_l"
      using exprTypeconInduct(4)[of True "fst cn" xe folded emptyTypedStore emptyStore
          emptyTypedStore e cd ?updState g e_l cd_l k_l m_l g'] uusDef by blast

     have cc1:"TypeSafe e_l (acc) (k_l) (m_l) (Storage (uus)) cd_l"
      unfolding TypeSafe_def
     proof intros
      show "AddressTypes acc" unfolding AddressTypes_def using tsE unfolding TypeSafe_def
AddressTypes_def using 1(7) transfer_type_same uusDef by simp
     next
      have trUus:"transfer (Address e) ?newAdd v' (Accounts uus) = Some acc"
       using 1(7) uusDef by simp
      have scUus':"safeContract (Accounts uus) (Storage uus)"
       using tsE unfolding TypeSafe_def by simp
      show "safeContract acc (state.Storage uus)"
      proof (unfold safeContract_def, intro allI impI)
       fix env ct' dud' i' tp
       assume in1:"Type (acc (Address (env::environment))) = Some (atype.Contract (Contract env)) ∧
                   ep $$ Contract env = Some (ct', dud') ∧
                   ct' $$ i' = Some (Var tp)"
       have typeOld:"Type (Accounts uus (Address env)) = Some (atype.Contract (Contract env))"
        using in1 transfer_type_same[OF trUus, of "Address env"] by simp
       show "SCon tp i' (state.Storage uus (Address env))"
        using scUus' in1 typeOld unfolding safeContract_def by blast
      qed
     next
      show "unique_locations (Denvalue e_l)" using exprTS unfolding TypeSafe_def by blast
     next
      show "compPointers k_l (Denvalue e_l)" using exprTS unfolding TypeSafe_def by blast
     next
      show "balanceTypes acc" using btACC by auto
     next
      have "envAddressesWellFormed e_l" using exprTS unfolding TypeSafe_def by blast
      then show "addressFormat (Address e_l)" and "addressFormat (Sender e_l)" by simp+
     next
      show "svalueTypes (Svalue e_l)" using exprTS unfolding TypeSafe_def by blast
     next
      show " lessThanTopLocs k_l" using exprTS unfolding TypeSafe_def by blast
     next
      show " lessThanTopLocs cd_l" using exprTS unfolding TypeSafe_def by blast
     next
      show "lessThanTopLocs m_l" using exprTS unfolding TypeSafe_def by blast
     next
      show "typeCompat (Denvalue e_l) k_l m_l (Storage uus (Address e_l)) cd_l" using uusDef exprTS un-
folding TypeSafe_def by blast
     next
      show "denvalueTypeCorrectness e_l k_l m_l" using TypeSafe_def exprTS by blast
     next
      show "subPrefixStructuralConsistency m_l" using TypeSafe_def exprTS by blast
     next
      show "SomeValSomeTyp m_l" using TypeSafe_def exprTS by blast
     qed

     have elFolded:"Address e_l = Address folded ∧ Contract e_l = Contract folded" using 1 a80
      using msel_ssel_expr_load_rexp_gas(4) aa1 by blast
     have cc2:"fullyInitialised e_l acc k_l"
     proof -
      have fiOld:"
       ∃c ct dud.
        Type (Accounts ?updState (Address e_l)) = Some (atype.Contract c) ∧
        Contract e_l = c ∧
        ep $$ c = Some (ct, dud) ∧
        (∀id v. (ct $$ id = Some (Var v)) = (Denvalue e_l $$ id = Some (type.Storage v, Storeloc
id))) ∧
```

```
            (∀ id v loc. Denvalue e_l $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
            (∀ t l p.
                (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_l) ∧ accessStore l k_l = Some (KStoptr
p) ⟶
                (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_l) ∧ CompStoType t' t l'
p))"
          using exprTS unfolding fullyInitialised_def by blast
        then obtain c_fi ct_fi dud_fi where fiC:"
            Type (Accounts ?updState (Address e_l)) = Some (atype.Contract c_fi) ∧
            Contract e_l = c_fi ∧
            ep $$ c_fi = Some (ct_fi, dud_fi) ∧
            (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e_l $$ id = Some (type.Storage v, Storeloc
id))) ∧
            (∀ id v loc. Denvalue e_l $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
          and fiPtr:"∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e_l) ∧ accessStore l k_l = Some (KStoptr
p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e_l) ∧ CompStoType t' t l' p)"
          by blast
        have typeNew:"Type (acc (Address e_l)) = Some (atype.Contract c_fi)"
          using fiC 1(7) transfer_type_same uusDef by simp
        show ?thesis
          unfolding fullyInitialised_def using fiC fiPtr typeNew by blast
      qed

      then show "TypeSafe e_l (Accounts (uus⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈))
(Stack
          (uus⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈))
     (Memory (uus⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈)) (Storage (uus⦇Gas:=g',Accounts
:=
          acc, Stack := k_l, Memory := m_l⦈)) cd_l
∧ fullyInitialised e_l acc k_l"
        using cc1 by auto
    qed
    have a140':"TypeSafe e_l (Accounts (?updState⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈))
        (Stack (?updState⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈))
   (Memory (?updState⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈)) (Storage
       (?updState⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈)) cd_l"
      using a140 uusDef by auto
    have "st⦇Gas := g', Accounts := (Accounts st)((?newAdd) := ⦇Bal = ShowL_int 0, Type = Some
(atype.Contract i), Contracts = 0⦈),
                                                Storage := (Storage st)((?newAdd) := {$$})⦈
= (?updState⦇Gas := g'⦈)" by simp
    then have a18Exp:"local.stmt (snd cn) e_l cd_l
     (?updState⦇Gas := g', Accounts := acc, Stack := k_l, Memory := m_l⦈) = Normal ((), st'')" using 1(8)
by argo
    have a150:"fullyInitialised e_l acc k_l" using a140 by simp


    have IH1:" StateInvariant e_l (?updState⦇Gas:=g',Accounts := acc, Stack := k_l, Memory := m_l⦈) st''
cd_l"
      using 11(1)[OF a10 a20 a30 a40 a50 a60 _ a65 a70 _ _ _ a80 a90 _ _ _ a110 a120 _ a130 a140'
a18Exp] a30 1(8) a150 by simp

    have st'Final:"st' = incrementAccountContracts (Address e) (st''⦇Stack := Stack st, Memory :=
Memory st⦈)" using 1 by blast

    then have sameStores:"Memory st = Memory st' ∧ Stack st = Stack st'" using 1 by simp
    show ?thesis unfolding StateInvariant_def
    proof intros
      show "TypeSafe e (Accounts st') (Stack st') (Memory st') (Storage st') cd" unfolding
TypeSafe_def
      proof intros
        have old:"AddressTypes (Accounts st'')" using IH1 unfolding StateInvariant_def TypeSafe_def by
```

```
auto
        show "AddressTypes (Accounts st')" using old unfolding AddressTypes_def using st'Final by
fastforce

      next
        have scOld:"safeContract (Accounts st'') (Storage st'')"
          using IH1 unfolding StateInvariant_def TypeSafe_def by blast
        show "safeContract (Accounts st') (Storage st')"
        proof (unfold safeContract_def, intro allI impI)
          fix env ct' dud' i' tp
          assume in1:"Type (Accounts st' (Address (env::environment))) = Some (atype.Contract (Contract
env)) ∧
                      ep $$ Contract env = Some (ct', dud') ∧ ct' $$ i' = Some (Var tp)"
          have typeOld:"Type (Accounts st'' (Address env)) = Some (atype.Contract (Contract env))"
            using in1 st'Final by simp
          have stoEq:"Storage st' (Address env) = Storage st'' (Address env)"
            using st'Final by simp
          have "SCon tp i' (Storage st'' (Address env))"
            using scOld in1 typeOld unfolding safeContract_def by blast
          then show "SCon tp i' (Storage st' (Address env))"
            using stoEq by simp
        qed
      next
        show "unique_locations (Denvalue e)" using 11(2) unfolding TypeSafe_def using sameStores by
simp
      next
        show "compPointers (Stack st') (Denvalue e)" using 11(2) st'Final unfolding StateInvariant_def
TypeSafe_def by simp
      next
        show "balanceTypes (Accounts st')" using IH1 unfolding StateInvariant_def TypeSafe_def
balanceTypes_def using st'Final sameStores by fastforce
      next
        have "envAddressesWellFormed e" using "11.prems"(1) unfolding TypeSafe_def by simp
        then show "addressFormat (Address e)" and "addressFormat (Sender e)" by simp+
      next
        show "svalueTypes (Svalue e)" using "11.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Stack st')" using 11(2) unfolding TypeSafe_def using sameStores by
simp
      next
        show "lessThanTopLocs cd" using "11.prems"(1) unfolding TypeSafe_def by simp
      next
        show "lessThanTopLocs (Memory st')" using 11(2) unfolding TypeSafe_def using sameStores by
simp
      next
        show "typeCompat (Denvalue e) (Stack st') (Memory st') (Storage st' (Address e)) cd"
          unfolding typeCompat_def
        proof intros
          fix t l
          assume inDen:"(t, l) |∈| fmran (Denvalue e)"

          have acc:"(∀adv. case Type (Accounts st adv) of None ⇒ True | Some EOA ⇒ True | Some
              (atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep)"
            using 11(2) unfolding TypeSafe_def AddressTypes_def by blast
          obtain cOld where cOldDef:"Type (Accounts st (Address e)) = Some (atype.Contract cOld) ∧
Contract e = cOld"
            using 11(4) unfolding fullyInitialised_def by simp
          then obtain ctO dudO where ctoDef: "ep $$ Contract e = Some (ctO, dudO)" using acc
            using "11.prems"(3) typesafe_base.fullyInitialised_def typesafe_base_axioms by blast
          have storageFinal:"Storage st' = Storage st''" using 1 by auto
          have scSt'':"safeContract (Accounts st'') (Storage st'')"
            using IH1 unfolding StateInvariant_def TypeSafe_def by blast
          have trUus:"transfer (Address e) ?newAdd v' (Accounts uus) = Some acc"
            using 1(7) uusDef by simp
```

```
          have typeStE:"Type (Accounts st (Address e)) = Some (atype.Contract (Contract e))"
            using cOldDef by simp
          have typeUusE:"Type (Accounts uus (Address e)) = Some (atype.Contract (Contract e))"
            using typeStE notSame uusDef by simp
          have typeAccE:"Type (acc (Address e)) = Some (atype.Contract (Contract e))"
            using transfer_type_same[OF trUus, of "Address e"] typeUusE by simp
          let ?stInit = "?updState⦇Gas := g', Accounts := acc, Stack := k_l, Memory := m_l⦈"
          have typeInitE:"Type (Accounts ?stInit (Address e))
                       = Some (atype.Contract (Contract e))"
            using typeAccE by simp
          have typeSt''E:"Type (Accounts st'' (Address e)) = Some (atype.Contract (Contract e))"
          proof (rule atype_same)
            show "local.stmt (snd cn) e_l cd_l ?stInit = Normal ((), st'')"
              using 1(8)
              using a18Exp by argo
            show "Type (Accounts ?stInit (Address e)) = Some (atype.Contract (Contract e))"
              using typeInitE .
          qed

          show "denvalueCon (Stack st') (Memory st') (Storage st' (Address e)) cd t l"
          proof(cases l)
            case (Stackloc x1)

            then show ?thesis
            proof(cases "accessStore x1 (Stack st')")
              case None
              then show ?thesis using 11(2) unfolding TypeSafe_def typeCompat_def using inDen
sameStores Stackloc by fastforce
            next
              case (Some a)
              then show ?thesis
              proof(cases a)
                case (KValue x1')
                then show ?thesis using 11(2) unfolding TypeSafe_def typeCompat_def using inDen
sameStores Stackloc Some by fastforce
              next
                case (KCDptr x2)
                then show ?thesis using 11(2) unfolding TypeSafe_def typeCompat_def using inDen
sameStores Stackloc Some by fastforce
              next
                case (KMemptr x3)
                then show ?thesis using 11(2) unfolding TypeSafe_def typeCompat_def using inDen
sameStores Stackloc Some by (cases t; fastforce)
              next
                case (KStoptr x4)
                then have accO:"accessStore x1 (Stack st) = Some( KStoptr x4)"
                  using Some 1 by auto
                then obtain struct where structDef: "t = type.Storage struct"
                  using 11(2) unfolding TypeSafe_def typeCompat_def using inDen Some KStoptr Stackloc
                  by (cases t) fastforce+
                obtain tprnt lprnt where tprntDef: "((type.Storage tprnt, Storeloc lprnt) |∈| fmran
(Denvalue e) ∧ CompStoType tprnt struct lprnt x4)"
                  using 11(4) using accO inDen unfolding fullyInitialised_def
                  using Stackloc structDef by blast
                then have "SCon tprnt lprnt (Storage st'' (Address e))"
                proof -
                  from tprntDef have inRan: "(type.Storage tprnt, Storeloc lprnt) |∈| fmran (Denvalue
e)" by simp
                  then obtain k where kDef: "Denvalue e $$ k = Some (type.Storage tprnt, Storeloc
lprnt)"
                    using fmlookup_ran_iff by auto
                  from 11(4) obtain ct_x dud_x where
                    epX: "ep $$ (Contract e) = Some (ct_x, dud_x)" and
                    bijX: "∀ id v. ct_x $$ id = Some (Var v) ⟷ Denvalue e $$ id = Some (type.Storage
```

```
v, Storeloc id)" and
                  uniqX: "∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id =
loc"
                  unfolding fullyInitialised_def by blast
                from kDef uniqX have "k = lprnt" by auto
                then have "Denvalue e $$ lprnt = Some (type.Storage tprnt, Storeloc lprnt)" using
kDef by simp
                then have "ct_x $$ lprnt = Some (Var tprnt)" using bijX by auto
                moreover from epX ctoDef have "ct_x = ct0" by auto
                ultimately have "ct0 $$ lprnt = Some (Var tprnt)" by simp
                then show ?thesis
                  using ctoDef scSt'' typeSt''E unfolding safeContract_def by blast
              qed

              then have "SCon struct x4 (Storage st'' (Address e))"
                using tprntDef SCon_imps_sublocs by blast
              then show ?thesis using Stackloc Some KStoptr structDef storageFinal by simp
            qed
          qed
        next
          case (Storeloc x2)
          then obtain struct where structDef: "t = type.Storage struct"
            using 11(2) inDen unfolding TypeSafe_def typeCompat_def by (cases t,fastforce+)

          then have "SCon struct x2 (Storage st'' (Address e))"
          proof -
            from inDen Storeloc structDef have inRan: "(type.Storage struct, Storeloc x2) |∈| fmran
(Denvalue e)" by simp
            then obtain k where kDef: "Denvalue e $$ k = Some (type.Storage struct, Storeloc x2)"
              using fmlookup_ran_iff by auto
            from 11(4) obtain ct_x dud_x where
              epX: "ep $$ (Contract e) = Some (ct_x, dud_x)" and
              bijX: "∀ id v. ct_x $$ id = Some (Var v) ⟷ Denvalue e $$ id = Some (type.Storage v,
Storeloc id)" and
              uniqX: "∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
              unfolding fullyInitialised_def by blast
            from kDef uniqX have "k = x2" by auto
            then have "Denvalue e $$ x2 = Some (type.Storage struct, Storeloc x2)" using kDef by
simp
            then have "ct_x $$ x2 = Some (Var struct)" using bijX by auto
            moreover from epX ctoDef have "ct_x = ct0" by auto
            ultimately have "ct0 $$ x2 = Some (Var struct)" by simp
            then show ?thesis
              using ctoDef scSt'' typeSt''E unfolding safeContract_def by blast
          qed
          then show ?thesis using Storeloc structDef
            by (simp add: storageFinal)
        qed
      qed
    next
      show "denvalueTypeCorrectness e (Stack st') (Memory st')" using 1(9) 11(2) unfolding
TypeSafe_def by simp
    next
      show "subPrefixStructuralConsistency (Memory st')" using 1(9) 11(2) unfolding TypeSafe_def by
simp
    next
      show "SomeValSomeTyp (Memory st')" using 1(9) 11(2) unfolding TypeSafe_def by simp
    qed
  next
    have fiOld:"
      ∃ c ct dud.
        Type (Accounts st (Address e)) = Some (atype.Contract c) ∧
        Contract e = c ∧
        ep $$ c = Some (ct, dud) ∧
```

```
          (∀ id v. (ct $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc id)))
∧
          (∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc) ∧
          (∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p))"
        using 11(4) unfolding fullyInitialised_def by blast
      then obtain c_fi ct_fi dud_fi where fiC:"
          Type (Accounts st (Address e)) = Some (atype.Contract c_fi) ∧
          Contract e = c_fi ∧
          ep $$ c_fi = Some (ct_fi, dud_fi) ∧
          (∀ id v. (ct_fi $$ id = Some (Var v)) = (Denvalue e $$ id = Some (type.Storage v, Storeloc
id))) ∧
          (∀ id v loc. Denvalue e $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc)"
        and fiPtr:"∀ t l p.
              (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st) = Some
(KStoptr p) ⟶
              (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
        by blast
      have typeNew:"Type (Accounts st' (Address e)) = Some (atype.Contract c_fi)"
        using IH1 fiC atype_same[OF 1(8)] st'Final unfolding StateInvariant_def
        using "11.prems"(2) atype_same by presburger
      have ptrNew:"∀ t l p.
          (type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') = Some
(KStoptr p) ⟶
          (∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t l' p)"
      proof (intro allI impI)
        fix t l p
        assume in1:"(type.Storage t, Stackloc l) |∈| fmran (Denvalue e) ∧ accessStore l (Stack st') =
Some (KStoptr p)"
        have "accessStore l (Stack st) = Some (KStoptr p)" using in1 sameStores by simp
        then show "∃ t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue e) ∧ CompStoType t' t
l' p"
          using fiPtr in1 by blast
      qed
      show "fullyInitialised e (Accounts st') (Stack st')"
        unfolding fullyInitialised_def using fiC typeNew ptrNew by blast

    next
      show "⋀locs t. accessTypeStore locs (Memory st) = Some t ⟹ accessTypeStore locs (Memory st') =
Some t"
        using sameStores by auto
    next
      show "⋀locs v.
       accessStore locs (Memory st) = Some (MPointer v) ⟹ ∃ v'. accessStore locs (Memory st') = Some
(MPointer v')"
        using sameStores by auto
    next
      show "⋀locs v. accessStore locs (Memory st) = Some (MValue v) ⟹ ∃ v'. accessStore locs (Memory
st') = Some (MValue v')"
        using sameStores by auto
    next
      show "⋀i loc.
       i < Toploc (Memory st) ∧ LSubPrefL2 loc (ShowL_nat i) ⟹
       accessStore loc (Memory st) = None ⟹ accessStore loc (Memory st') = None"
        using sameStores by auto
    next
      show "Toploc (Memory st) ≤ Toploc (Memory st')"
        using sameStores by auto
    qed
  qed
qed
```

end

end

18 Examples (TypeSafe_Examples)

theory *TypeSafe_Examples*
  imports *TypeSafe*
begin

18.1 Satisfiability of the locale assumptions

The assumptions of *typesafe_base* hold for the empty procedure environment and for a non-empty one.

global_interpretation *typesafe_vacuous: typesafe_base costs_ex fmempty costs_min*
  by *unfold_locales auto*

Non-vacuous sanity check: a concrete non-empty procedure environment.

definition *nv_ct :: "(Identifier, member) fmap"*
  where *"nv_ct =*
    fmap_of_list
      [(STR ''x'', Var (STValue TBool)),
       (STR ''y'', Var (STValue (TUInt b8)))]"

definition *nv_ctor :: "(Identifier × type) list × s"*
  where *"nv_ctor = ([], SKIP)"*

definition *nv_contract :: contract*
  where *"nv_contract = (nv_ct, nv_ctor, SKIP)"*

definition *ep_nonvacuous :: environment$_p$*
  where *"ep_nonvacuous = fmap_of_list [(STR ''C'', nv_contract)]"*

lemma *ep_nonvacuous_has_two_vars:*
  "∃ct dud t1 t2.
     ep_nonvacuous $$ (STR ''C'') = Some (ct, dud) ∧
     ct $$ (STR ''x'') = Some (Var t1) ∧
     ct $$ (STR ''y'') = Some (Var t2) ∧
     (STR ''x'') ≠ (STR ''y'')"
  unfolding *ep_nonvacuous_def nv_contract_def nv_ct_def nv_ctor_def*
  by *simp*

global_interpretation *typesafe_nonvacuous: typesafe_base costs_ex ep_nonvacuous costs_min*
  by *(unfold_locales; auto simp: ep_nonvacuous_def nv_contract_def nv_ct_def nv_ctor_def)*

18.2 Memory traversal example

abbreviation *mymemory22::memoryT*
  where *"mymemory22 ≡*
    ⦇Mapping = fmap_of_list
      [(STR ''1.1.0'', MValue STR ''False''),
       (STR ''0.1.0'', MValue STR ''True''),
       (STR ''1.0'', MPointer STR ''2.0''),
       (STR ''1.1.0.0'', MValue STR ''False''),
       (STR ''0.2.0'', MValue STR ''True''),
       (STR ''0.0'', MPointer STR ''0.0'')],
     Toploc = 1,
     Typed_Mapping = fmap_of_list
      [(STR ''1.1.0'', MTValue TBool),
       (STR ''0.1.0'', MTValue TBool),
       (STR ''1.0'', MTArray 2 (MTValue TBool)),
       (STR ''1.1.0.0'', MTValue TBool),
       (STR ''0.2.0'', MTValue TBool),

```
      (STR ''0.0'', MTArray 2 (MTValue TBool))]⦈"
lemma mymemory22_locations:
  "GetAllMemoryLocations (MTArray 3 (MTArray 2 (MTValue TBool))) STR ''0'' mymemory22 =
     {STR ''0.2.0'', STR ''1.0'', STR ''1.0.0'', STR ''0.0''}"
  by eval
```

18.3 Concrete preservation examples

Concrete usability demo for *typesafe_base.TypeSafe_Statements* with a non-empty contract environment.

The setup models the following Solidity contract:

```
contract C {
    bool x;
    uint8 y;

    function ping() public {
        // SKIP
    }
}
```

- contract identifier *STR ''C''* is looked up in *ep_nonvacuous*;
- storage variables *STR ''x''* and *STR ''y''* map to *Var (STValue TBool)* and *Var (STValue (TUInt b8))* in *nv_ct*;
- the environment is produced by *ffold_init*, so its *Denvalue* has *Storeloc* entries;
- statement *SKIP* plays the role of an empty function body.

```
definition demo_addr :: address
  where "demo_addr = createAddress (STR ''A'')"

definition demo_sender :: address
  where "demo_sender = createAddress (STR ''S'')"

definition demo_svalue :: valuetype
  where "demo_svalue = createUInt b256 0"

definition demo_env :: environment
  where "demo_env =
    ffold_init nv_ct (emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue) (fmdom nv_ct)"

definition demo_acc :: accounts
  where "demo_acc =
    emptyAccount(demo_addr := emptyAcc⦇Type := Some (atype.Contract (STR ''C''))⦈)"

definition demo_state :: state
  where "demo_state =
    ⦇Accounts = demo_acc,
     Stack = emptyStore,
     Memory = emptyTypedStore,
     Storage = emptyStorage,
     Gas = 1⦈"

definition demo_cd :: calldataT
  where "demo_cd = emptyTypedStore"

lemma demo_safeContract_base:
  "typesafe_base.safeContract ep_nonvacuous demo_acc emptyStorage"
proof (unfold typesafe_nonvacuous.safeContract_def, intro allI impI)
  fix e ct dud i tp
  assume h:
    "Type (demo_acc (Address (e::environment))) = Some (atype.Contract (Contract e)) ∧
     ep_nonvacuous $$ environment.Contract (e::environment) = Some (ct, dud) ∧
```

```
     fmlookup ct i = Some (Var tp)"

  have tc_ival: "⋀t. typeCon t (ival t)"
  proof -
    fix t
    show "typeCon t (ival t)"
      using ivalTypeCon[of t "ival t"] by simp
  qed

  from h have ct_def: "ct = nv_ct"
    unfolding ep_nonvacuous_def nv_contract_def
    by (simp split:if_splits)
  from h ct_def consider
     (1) "(i = STR ''x'' ∧ tp = STValue TBool)"
    | (2) "(i = STR ''y'' ∧ tp = STValue (TUInt b8))"
    unfolding nv_ct_def
    by (auto split: if_splits member.splits)

  then show "SCon tp i (emptyStorage (Address e))"
  proof (cases)
    case 1
    then show ?thesis
      unfolding emptyStorage_def accessStorage_def
      using tc_ival
      by (metis SCon.simps(1) accessStorage_def fmempty_lookup option.case(1))
  next
    case 2
    then show ?thesis
      unfolding emptyStorage_def accessStorage_def
      using tc_ival by (metis SCon.simps(1) accessStorage_def fmempty_lookup option.case(1))
  qed
qed

lemma demo_safeContract:
  "typesafe_nonvacuous.safeContract demo_acc emptyStorage"
  using demo_safeContract_base
  by (simp add: typesafe_nonvacuous.safeContract_def)

lemma demo_addressFormat_addr: "addressFormat demo_addr"
  unfolding addressFormat_def typeCon.simps checkAddress_def demo_addr_def
  using createAddressNoDots by blast

lemma demo_addressFormat_sender: "addressFormat demo_sender"
  unfolding addressFormat_def typeCon.simps checkAddress_def demo_sender_def
  using createAddressNoDots by blast

lemma demo_balanceTypes: "balanceTypes demo_acc"
  unfolding balanceTypes_def demo_acc_def emptyAccount_def emptyAcc_def
  by (simp add: checkUInt_def, eval)

lemma demo_AddressTypes: "typesafe_base.AddressTypes ep_nonvacuous demo_acc"
proof -
  have c_dom: "STR ''C'' |∈| fmdom ep_nonvacuous"
    unfolding ep_nonvacuous_def by simp
  have type_cases:
    "Type (demo_acc adv) =
     (if adv = demo_addr then Some (atype.Contract (STR ''C'')) else None)" for adv
    unfolding demo_acc_def emptyAccount_def emptyAcc_def
    by simp
  have case_eval:
    "case Type (demo_acc adv) of
       Some (atype.Contract c) ⇒ addressFormat adv ∧ c |∈| fmdom ep_nonvacuous
     | _ ⇒ True =
     (if adv = demo_addr then addressFormat adv ∧ STR ''C'' |∈| fmdom ep_nonvacuous else True)" for adv
```

```
    using type_cases
    using c_dom demo_addressFormat_addr by auto
  show ?thesis
    unfolding typesafe_nonvacuous.AddressTypes_def
  proof (intros)
    fix adv
    show "case Type (demo_acc adv) of None ⇒ True | Some EOA ⇒ True | Some (atype.Contract c) ⇒
addressFormat adv ∧ c |∈| fmdom ep_nonvacuous"
      using case_eval demo_addressFormat_addr c_dom
      by (simp add: type_cases)
  qed
qed

lemma demo_typeSafe:
  "typesafe_base.TypeSafe ep_nonvacuous demo_env (Accounts demo_state) (Stack demo_state) (Memory
demo_state) (Storage demo_state) demo_cd"
proof -
  have sc: "typesafe_base.safeContract ep_nonvacuous (Accounts demo_state) (Storage demo_state)"
    using demo_safeContract_base unfolding demo_state_def by simp
  have bt: "balanceTypes (Accounts demo_state)"
    using demo_balanceTypes unfolding demo_state_def by simp
  have sv: "svalueTypes demo_svalue"
    unfolding demo_svalue_def svalueTypes_def typeCon.simps checkUInt_def createUInt_def by eval
  have ltp: "lessThanTopLocs emptyTypedStore"
    using typesafe_nonvacuous.typedEmptyTopLocs by simp
  have ep_lookup:
    "ep_nonvacuous $$ Contract (emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue)
      = Some (nv_ct, (nv_ctor, SKIP))"
    unfolding ep_nonvacuous_def nv_contract_def by simp
  have af1: "addressFormat demo_addr" using demo_addressFormat_addr .
  have af2: "addressFormat demo_sender" using demo_addressFormat_sender .
  have at: "typesafe_nonvacuous.AddressTypes demo_acc" using demo_AddressTypes .
  have sp: "typesafe_nonvacuous.subPrefixStructuralConsistency emptyTypedStore"
    unfolding typesafe_nonvacuous.subPrefixStructuralConsistency_def accessTypeStore_def
emptyTypedStore_def by simp
  have svt: "typesafe_nonvacuous.SomeValSomeTyp emptyTypedStore"
    unfolding typesafe_nonvacuous.SomeValSomeTyp_def accessStore_def accessTypeStore_def
emptyTypedStore_def by simp
  have coh: "Type (demo_acc demo_addr) = Some (atype.Contract (STR ''C''))"
    unfolding demo_state_def demo_acc_def demo_env_def ffold_init_def emptyAccount_def emptyAcc_def by
simp
  have initTS:
    "∀ e' x y.
      ffold (init nv_ct) (emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue) (fmdom nv_ct) = e'
⟶
      typesafe_base.TypeSafe ep_nonvacuous e' demo_acc emptyStore emptyTypedStore emptyStorage
emptyTypedStore ∧
      (Denvalue e' $$ x = Some y ⟶ snd y = Storeloc x) ∧
      (Denvalue e' $$ x = Some y ⟶ (∃ t1. nv_ct $$ x = Some (Var t1) ∧ fst y = type.Storage t1))"
    using typesafe_nonvacuous.ffoldInitTypeSafe[OF sc bt sv ltp ep_lookup af1 af2 ] at demo_state_def
    by (metis coh emptyEnv_members sp state.select_convs(1,4) svt)
  then show ?thesis
    unfolding demo_env_def demo_state_def demo_cd_def ffold_init_def
    by simp
qed

lemma demo_fullyInitialised:
  "typesafe_base.fullyInitialised ep_nonvacuous demo_env (Accounts demo_state) (Stack demo_state)"
proof -
  have cfi:
    "Type (Accounts demo_state (Address demo_env)) = Some (atype.Contract (STR ''C'')) ∧ Contract
demo_env = (STR ''C'')"
    unfolding demo_state_def demo_acc_def demo_env_def ffold_init_def emptyAccount_def emptyAcc_def
    by simp
```

```
  have epfi: "ep_nonvacuous $$ Contract demo_env = Some (nv_ct, (nv_ctor, SKIP))"
    unfolding demo_env_def ffold_init_def ep_nonvacuous_def nv_contract_def by simp
  have mapfi:
    "∀id v. (nv_ct $$ id = Some (Var v)) = (Denvalue demo_env $$ id = Some (type.Storage v, Storeloc
id))"
    using typesafe_nonvacuous.ffoldInit_var_storage_mapping_eq[of nv_ct demo_addr "STR ''C''"
demo_sender demo_svalue demo_env]
    unfolding demo_env_def ffold_init_def by simp
  have locfi:
    "∀id v loc. Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
  proof (intro allI impI)
    fix id v loc
    assume h: "Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc)"
    have hloc:
      "Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v, Storeloc
loc) = Storeloc id"
      using typesafe_nonvacuous.ffoldInitAllLocsStorage[
          of nv_ct
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             demo_svalue
             "fmdom nv_ct"]
      unfolding demo_env_def ffold_init_def by simp
    then show "id = loc" using h by simp
  qed
  have ptrfi:
    "∀t l p.
      (type.Storage t, Stackloc l) |∈| fmran (Denvalue demo_env) ∧
      accessStore l (Stack demo_state) = Some (KStoptr p) ⟶
      (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue demo_env) ∧ CompStoType t' t l' p)"
    unfolding demo_state_def by (simp add: accessStore_def emptyStore_def)
  show ?thesis
    unfolding typesafe_base.fullyInitialised_def demo_state_def
    using cfi epfi mapfi locfi ptrfi
    by (simp add: demo_state_def typesafe_nonvacuous.fullyInitialised_def)
qed
```

Concrete non-vacuous witness for the pointer conjunct in `typesafe_base.fullyInitialised_def`. Unlike `demo_fullyInitialised`, this setup includes a storage-typed stack binding that points to an existing storage location, so the pointer implication is exercised with a true antecedent.

```
definition demo_ptr_id :: Identifier
  where "demo_ptr_id = STR ''ptr_tmp''"

definition demo_ptr_stack_loc :: location
  where "demo_ptr_stack_loc = STR ''0''"

definition demo_ptr_target_loc :: location
  where "demo_ptr_target_loc = STR ''x''"

definition demo_env_with_stptr :: environment
  where "demo_env_with_stptr =
    demo_env(|Denvalue := fmupd demo_ptr_id
      (type.Storage (STValue TBool), Stackloc demo_ptr_stack_loc) (Denvalue demo_env)|)"

definition demo_state_with_stptr :: state
  where "demo_state_with_stptr =
    demo_state(|Stack := updateStore demo_ptr_stack_loc (KStoptr demo_ptr_target_loc) (Stack
demo_state)|)"

lemma demo_pointer_clause_nonvacuous:
  "∃t l p.
      (type.Storage t, Stackloc l) |∈| fmran (Denvalue demo_env_with_stptr) ∧
      accessStore l (Stack demo_state_with_stptr) = Some (KStoptr p) ∧
```

```
      (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue demo_env_with_stptr) ∧ CompStoType t'
t l' p)"
proof -
  have mapfi:
    "∀id v. (nv_ct $$ id = Some (Var v)) = (Denvalue demo_env $$ id = Some (type.Storage v, Storeloc
id))"
    using typesafe_nonvacuous.ffoldInit_var_storage_mapping_eq[of nv_ct demo_addr "STR ''C''"
demo_sender demo_svalue demo_env]
    unfolding demo_env_def ffold_init_def by simp
  have den_target:
    "Denvalue demo_env $$ demo_ptr_target_loc =
      Some (type.Storage (STValue TBool), Storeloc demo_ptr_target_loc)"
    unfolding demo_ptr_target_loc_def using mapfi nv_ct_def
    by (metis fmap_of_list_simps(2) fmupd_lookup)
  have den_ptr:
    "Denvalue demo_env_with_stptr $$ demo_ptr_id =
      Some (type.Storage (STValue TBool), Stackloc demo_ptr_stack_loc)"
    unfolding demo_env_with_stptr_def by simp
  have den_target':
    "Denvalue demo_env_with_stptr $$ demo_ptr_target_loc =
      Some (type.Storage (STValue TBool), Storeloc demo_ptr_target_loc)"
    unfolding demo_env_with_stptr_def using den_target
    by (simp add: demo_ptr_id_def demo_ptr_target_loc_def)
  have in_stack:
    "(type.Storage (STValue TBool), Stackloc demo_ptr_stack_loc) |∈| fmran (Denvalue
demo_env_with_stptr)"
    using den_ptr by (simp add: fmranI)
  have in_store:
    "(type.Storage (STValue TBool), Storeloc demo_ptr_target_loc) |∈| fmran (Denvalue
demo_env_with_stptr)"
    using den_target' by (simp add: fmranI)
  have ptr_stack:
    "accessStore demo_ptr_stack_loc (Stack demo_state_with_stptr) = Some (KStoptr demo_ptr_target_loc)"
    unfolding demo_state_with_stptr_def by simp
  have comp_self:
    "CompStoType (STValue TBool) (STValue TBool) demo_ptr_target_loc demo_ptr_target_loc"
    using CompStoType_sameLocNdTyp by simp
  show ?thesis
    using in_stack in_store ptr_stack comp_self by blast
qed
```

Concrete 3D memory-array witness aligned with `decl` initialisation: an outer array of length 2 whose element type is a 2D bool array.

```
definition demo_mem3d_elem_type :: mtypes
  where "demo_mem3d_elem_type = MTArray 2 (MTArray 2 (MTValue TBool))"

definition demo_mem3d_type :: mtypes
  where "demo_mem3d_type = MTArray 2 demo_mem3d_elem_type"

definition demo_mem3d_root :: location
  where "demo_mem3d_root = ShowL_nat (Toploc (Memory demo_state))"

definition demo_mem3d :: memoryT
  where "demo_mem3d = minit 2 demo_mem3d_elem_type (Memory demo_state)"

definition demo_mem3d_id :: Identifier
  where "demo_mem3d_id = STR ''m3d''"

definition demo_mem3d_stack_loc :: location
  where "demo_mem3d_stack_loc = ShowL_nat (Toploc (Stack demo_state))"

definition demo_stack_mem3d :: stack
  where "demo_stack_mem3d = push (KMemptr demo_mem3d_root) (Stack demo_state)"
```

```
definition demo_env_mem3d :: environment
  where "demo_env_mem3d =
    updateEnv demo_mem3d_id (type.Memory demo_mem3d_type) (Stackloc demo_mem3d_stack_loc) demo_env"

definition demo_state_mem3d :: state
  where "demo_state_mem3d = demo_state(|Stack := demo_stack_mem3d, Memory := demo_mem3d|)"

lemma demo_mem3d_id_fresh:
  "Denvalue demo_env $$ demo_mem3d_id = None"
proof (rule ccontr)
  assume h: "Denvalue demo_env $$ demo_mem3d_id ≠ None"
  then obtain x where hx: "Denvalue demo_env $$ demo_mem3d_id = Some x"
    by (cases "Denvalue demo_env $$ demo_mem3d_id") auto
  have in_dom: "demo_mem3d_id |∈| fmdom nv_ct"
  proof -
    have imp:
      "Denvalue demo_env $$ demo_mem3d_id = Some x ⟶ demo_mem3d_id |∈| fmdom nv_ct"
      using ffold_init_emptyDen[of nv_ct demo_addr "STR ''C''" demo_sender demo_svalue
          "fmdom nv_ct" demo_mem3d_id x]
      unfolding demo_env_def ffold_init_def .
    show ?thesis
      using imp hx by blast
  qed
  show False
    using in_dom
    unfolding demo_mem3d_id_def nv_ct_def
    by simp
qed

lemma demo_decl_mem3d_matches:
  "decl demo_mem3d_id (type.Memory demo_mem3d_type) None False
      demo_cd (Memory demo_state) (Storage demo_state (Address demo_env))
      (demo_cd, Memory demo_state, Stack demo_state, demo_env) =
   Some (demo_cd, demo_mem3d, demo_stack_mem3d, demo_env_mem3d)"
proof -
  have fresh: "Denvalue demo_env $$ demo_mem3d_id = None"
    using demo_mem3d_id_fresh .
  have arrs: "arraysGreaterZero demo_mem3d_type"
    unfolding demo_mem3d_type_def demo_mem3d_elem_type_def by simp
  have stk_env:
    "astack_dup demo_mem3d_id (type.Memory demo_mem3d_type) (KMemptr demo_mem3d_root)
      (Stack demo_state, demo_env) = (demo_stack_mem3d, demo_env_mem3d)"
  proof -
    have "astack_dup demo_mem3d_id (type.Memory demo_mem3d_type) (KMemptr demo_mem3d_root)
        (Stack demo_state, demo_env) =
      astack demo_mem3d_id (type.Memory demo_mem3d_type) (KMemptr demo_mem3d_root)
        (Stack demo_state, demo_env)"
      using astack_dup_is_astack[OF fresh] by auto
    also have "... = (demo_stack_mem3d, demo_env_mem3d)"
      unfolding demo_stack_mem3d_def demo_env_mem3d_def
                demo_mem3d_stack_loc_def demo_mem3d_root_def
      by simp
    finally show ?thesis .
  qed
  show ?thesis
    unfolding demo_mem3d_id_def demo_mem3d_type_def demo_mem3d_elem_type_def
              demo_cd_def demo_state_def demo_mem3d_def demo_stack_mem3d_def
              demo_env_mem3d_def demo_mem3d_root_def demo_mem3d_stack_loc_def
    using fresh arrs stk_env
    by (simp add: decl.simps(18) demo_mem3d_id_def)
qed

lemma demo_mem3d_root_empty:
  "accessStore demo_mem3d_root demo_mem3d = None"
```

```
  unfolding demo_mem3d_def demo_mem3d_elem_type_def demo_mem3d_root_def demo_state_def
  by eval

lemma demo_stack_mem3d_pointer:
  "accessStore demo_mem3d_stack_loc demo_stack_mem3d = Some (KMemptr demo_mem3d_root)"
  unfolding demo_mem3d_stack_loc_def demo_stack_mem3d_def demo_state_def accessStore_def
  by (metis Option.option.simps(3) accessStore_def decl_env_memory demo_decl_mem3d_matches
demo_mem3d_id_fresh demo_mem3d_root_def demo_stack_mem3d_def
      demo_state_def)

lemma demo_env_mem3d_binding:
  "Denvalue demo_env_mem3d $$ demo_mem3d_id =
    Some (type.Memory demo_mem3d_type, Stackloc demo_mem3d_stack_loc)"
  unfolding demo_env_mem3d_def
  by simp


lemma demo_typeSafe_mem3d:
  "typesafe_base.TypeSafe ep_nonvacuous demo_env_mem3d
    (Accounts demo_state_mem3d) (Stack demo_state_mem3d) (Memory demo_state_mem3d) (Storage
demo_state_mem3d) demo_cd"
proof -
  have base_ts:
    "typesafe_base.TypeSafe ep_nonvacuous demo_env
      (Accounts demo_state) (Stack demo_state) (Memory demo_state) (Storage demo_state) demo_cd"
    using demo_typeSafe .
  have decl_ok:
    "decl demo_mem3d_id (type.Memory demo_mem3d_type) None False
        demo_cd (Memory demo_state) (Storage demo_state (Address demo_env))
        (demo_cd, Memory demo_state, Stack demo_state, demo_env) =
     Some (demo_cd, demo_mem3d, demo_stack_mem3d, demo_env_mem3d)"
    using demo_decl_mem3d_matches .
  have step:
    "typesafe_base.TypeSafe ep_nonvacuous demo_env_mem3d
      (Accounts demo_state) demo_stack_mem3d demo_mem3d (Storage demo_state) demo_cd"
    using typesafe_nonvacuous.typeSafeDeclNone[OF base_ts decl_ok]
    by blast
  show ?thesis
    using step
    unfolding demo_state_mem3d_def
    by simp
qed

lemma demo_mem3d_lessThanTopLocs:
  "lessThanTopLocs demo_mem3d"
  using demo_typeSafe_mem3d
  unfolding typesafe_nonvacuous.TypeSafe_def demo_state_mem3d_def
  by simp

lemma demo_mem3d_subPrefixStructuralConsistency:
  "typesafe_nonvacuous.subPrefixStructuralConsistency demo_mem3d"
  using demo_typeSafe_mem3d
  unfolding typesafe_nonvacuous.TypeSafe_def demo_state_mem3d_def
  by simp

lemma demo_mem3d_SomeValSomeTyp:
  "typesafe_nonvacuous.SomeValSomeTyp demo_mem3d"
  using demo_typeSafe_mem3d
  unfolding typesafe_nonvacuous.TypeSafe_def demo_state_mem3d_def
  by simp

lemma demo_fullyInitialised_mem3d:
  "typesafe_base.fullyInitialised ep_nonvacuous demo_env_mem3d (Accounts demo_state_mem3d) (Stack
demo_state_mem3d)"
```

```
proof -
  have cfi:
    "Type (Accounts demo_state_mem3d (Address demo_env_mem3d)) = Some (atype.Contract (STR ''C'')) ∧
     Contract demo_env_mem3d = (STR ''C'')"
    unfolding demo_state_mem3d_def demo_state_def demo_acc_def demo_env_mem3d_def
              demo_env_def ffold_init_def emptyAccount_def emptyAcc_def
    by simp
  have epfi:
    "ep_nonvacuous $$ Contract demo_env_mem3d = Some (nv_ct, (nv_ctor, SKIP))"
    unfolding demo_env_mem3d_def demo_env_def ffold_init_def ep_nonvacuous_def nv_contract_def
    by simp
  have mapfi_base:
    "∀id v. (nv_ct $$ id = Some (Var v)) = (Denvalue demo_env $$ id = Some (type.Storage v, Storeloc
id))"
    using typesafe_nonvacuous.ffoldInit_var_storage_mapping_eq[of nv_ct demo_addr "STR ''C''"
demo_sender demo_svalue demo_env]
    unfolding demo_env_def ffold_init_def by simp
  have mapfi:
    "∀id v. (nv_ct $$ id = Some (Var v)) = (Denvalue demo_env_mem3d $$ id = Some (type.Storage v,
Storeloc id))"
    using mapfi_base
    unfolding demo_env_mem3d_def demo_mem3d_id_def nv_ct_def
    by force
  have locfi_base:
    "∀id v loc. Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
  proof (intro allI impI)
    fix id v loc
    assume h: "Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc)"
    have hloc:
      "Denvalue demo_env $$ id = Some (type.Storage v, Storeloc loc) ⟶ snd (type.Storage v, Storeloc
loc) = Storeloc id"
      using typesafe_nonvacuous.ffoldInitAllLocsStorage[
          of nv_ct
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             "emptyEnv demo_addr (STR ''C'') demo_sender demo_svalue"
             demo_svalue
             "fmdom nv_ct"]
      unfolding demo_env_def ffold_init_def by simp
    then show "id = loc" using h by simp
  qed
  have locfi:
    "∀id v loc. Denvalue demo_env_mem3d $$ id = Some (type.Storage v, Storeloc loc) ⟶ id = loc"
    using locfi_base
    unfolding demo_env_mem3d_def demo_mem3d_id_def
    by auto
  have ptrfi:
    "∀t l p.
      (type.Storage t, Stackloc l) |∈| fmran (Denvalue demo_env_mem3d) ∧
      accessStore l (Stack demo_state_mem3d) = Some (KStoptr p) ⟶
      (∃t' l'. (type.Storage t', Storeloc l') |∈| fmran (Denvalue demo_env_mem3d) ∧ CompStoType t' t l'
p)"
    unfolding demo_env_mem3d_def demo_state_mem3d_def demo_stack_mem3d_def demo_state_def
    by (simp add: accessStore_def emptyStore_def push_def allocate_def updateStore_def)
  show ?thesis
    unfolding typesafe_base.fullyInitialised_def typesafe_nonvacuous.fullyInitialised_def
    using cfi epfi mapfi locfi ptrfi
    by auto
qed

lemma demo_stmt_normal:
  "typesafe_nonvacuous.stmt SKIP demo_env demo_cd demo_state = Normal ((), demo_state)"
  unfolding demo_state_def demo_cd_def
  by (simp add: typesafe_nonvacuous.stmt.simps demo_state_def demo_cd_def)
```

```
lemma demo_stmt_normal_mem3d:
  "typesafe_nonvacuous.stmt SKIP demo_env_mem3d demo_cd demo_state_mem3d = Normal ((),
demo_state_mem3d)"
  unfolding demo_env_mem3d_def demo_state_mem3d_def demo_state_def demo_cd_def
  by (simp add: typesafe_nonvacuous.stmt.simps)
```

Applying `typesafe_base.TypeSafe_Statements` to both concrete states.

```
lemma demo_StateInvariant_from_TypeSafe_Statements:
  "typesafe_base.StateInvariant ep_nonvacuous demo_env demo_state demo_state demo_cd"
  using typesafe_nonvacuous.TypeSafe_Statements[OF demo_typeSafe demo_stmt_normal
demo_fullyInitialised] .

lemma demo_mem3d_StateInvariant_from_TypeSafe_Statements:
  "typesafe_base.StateInvariant ep_nonvacuous demo_env_mem3d demo_state_mem3d demo_state_mem3d demo_cd"
  using typesafe_nonvacuous.TypeSafe_Statements[OF demo_typeSafe_mem3d demo_stmt_normal_mem3d
      demo_fullyInitialised_mem3d] .
```

Evaluating the memory example: the stack holds a reference to the outer array at location `0`, whose first element is a self-pointing inner array.

```
lemma demo_mem3d_eval:
  "demo_mem3d_root = STR ''0''"
  "accessStore demo_mem3d_stack_loc (Stack demo_state_mem3d) = Some (KMemptr (STR ''0''))"
  "accessStore (hash demo_mem3d_root (STR ''0'')) (Memory demo_state_mem3d) =
     Some (MPointer (STR ''0.0''))"
  by eval+

end
```

References

[1] Diego Marmsoler and Achim D. Brucker. Isabelle/Solidity: A deep embedding of Solidity in Isabelle/HOL. *Archive of Formal Proofs*, July 2022. https://isa-afp.org/entries/Solidity.html, Formal proof development.

[2] Billy Thornton and Diego Marmsoler. Type safety for Isabelle/Solidity. In Chutiporn Anutariya and Marcello M. Bonsangue, editors, *Theoretical Aspects of Computing - ICTAC 2024 - 21st International Colloquium, Bangkok, Thailand, November 25-29, 2024, Proceedings*, volume 15373 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2024.

[3] Billy Thornton and Diego Marmsoler. Type safety for Isabelle/Solidity. *Theoretical Computer Science*, 1067:115632, 2026.