

Unbounded Separation Logic

Thibault Dardinier

Department of Computer Science, ETH Zurich, Switzerland

February 6, 2026

Abstract

Many separation logics [11] support fractional permissions [3, 2] to distinguish between read and write access to a heap location, for instance, to allow concurrent reads while enforcing exclusive writes. Fractional permissions extend to composite assertions such as (co)inductive predicates and magic wands by allowing those to be multiplied [8, 4, 6] by a fraction. Typical separation logic proofs require that this multiplication has three key properties: it needs to distribute over assertions, it should permit fractions to be factored out from assertions, and two fractions of the same assertion should be combinable into one larger fraction.

Existing formal semantics incorporating fractional assertions into a separation logic define multiplication semantically (via models), resulting in a semantics in which distributivity and combinability do not hold for key resource assertions such as magic wands, and fractions cannot be factored out from a separating conjunction. By contrast, existing automatic separation logic verifiers [9, 7, 10, 1] define multiplication syntactically, resulting in a different semantics for which it is unknown whether distributivity and combinability hold for all assertions.

In this entry, we present and formalize an *unbounded* version of separation logic [5], a novel semantics for separation logic assertions that allows states to hold more than a full permission to a heap location during the evaluation of an assertion. By reimposing upper bounds on the permissions held per location at statement boundaries, we retain key properties of separation logic, in particular, we prove that the frame rule still holds. We also prove that our assertion semantics unifies semantic and syntactic multiplication and thereby reconciles the discrepancy between separation logic theory and tools and enjoys distributivity, factorisability, and combinability.

Contents

1	Unbounded Separation Logic	3
1.1	Assertions and state model	3
1.2	Useful lemmas	5

2	Frame rule	8
3	Distributivity and Factorisability	10
3.1	DotPos	10
3.2	DotDot	10
3.3	DotStar	12
3.4	DotWand	13
3.5	DotOr	14
3.6	DotAnd	15
3.7	DotImp	15
3.8	DotPure	16
3.9	DotFull	17
3.10	DotExists	17
3.11	DotForall	18
3.12	Split	18
4	Combinability	19
5	(Co)Inductive Predicates	26
5.1	Definitions	26
5.2	Everything preserves monotonicity	28
5.2.1	Monotonicity	28
5.2.2	Non-increasing	32
5.3	Tarski's fixed points	36
5.3.1	Greatest Fixed Point	36
5.3.2	Least Fixed Point	37
5.4	Combinability and (an assertion being) intuitionistic are set-closure properties	37
5.4.1	Intuitionistic assertions	37
5.4.2	Combinable assertions	39
5.5	Transfinite induction	41
5.6	Theorems	46
5.6.1	Greatest Fixed Point	46
5.6.2	Least Fixed Point	47
6	Properties of Magic Wands	48
7	Fractional Predicates and Magic Wands in Automatic Separation Logic Verifiers	51
7.1	Syntactic multiplication	51
7.2	Monotonicity and fixed point	55
7.3	Combinability	56
7.4	Theorems	58

1 Unbounded Separation Logic

```
theory UnboundedLogic
  imports Main
begin
```

1.1 Assertions and state model

We define our assertion language as described in Section 2.3 of the paper [5].

```
datatype ('a, 'b, 'c, 'd) assertion =
  Sem ('d  $\Rightarrow$  'c)  $\Rightarrow$  'a  $\Rightarrow$  bool
  | Mult 'b ('a, 'b, 'c, 'd) assertion
  | Star ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Wand ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Or ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | And ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Imp ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Exists 'd ('a, 'b, 'c, 'd) assertion
  | Forall 'd ('a, 'b, 'c, 'd) assertion
  | Pred
  | Bounded ('a, 'b, 'c, 'd) assertion
  | Wildcard ('a, 'b, 'c, 'd) assertion

type-synonym 'a command = ('a  $\times$  'a option) set

locale pre-logic =
  fixes plus :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a option (infixl <math>\oplus\Rightarrow 'a  $\Rightarrow$  bool (infixl <math>\langle\#\#\rangle 60) where
  a  $\langle\#\#\rangle$  b  $\longleftrightarrow$  a  $\oplus$  b  $\neq$  None

definition larger :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infixl <math>\langle\succ\rangle 55) where
  a  $\langle\succ\rangle$  b  $\longleftrightarrow$  ( $\exists$  c. Some a = b  $\oplus$  c)

end

type-synonym ('a, 'b, 'c) interp = ('a  $\Rightarrow$  'b)  $\Rightarrow$  'c set
```

The following locale captures the state model described in Section 2.2 of the paper [5].

```
locale logic = pre-logic +

  fixes mult :: 'b  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl <math>\langle\odot\rangle 64)

  fixes smult :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b
  fixes sadd :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b
  fixes sinv :: 'b  $\Rightarrow$  'b
```

fixes *one* :: 'b

fixes *valid* :: 'a \Rightarrow bool

assumes *commutative*: $a \oplus b = b \oplus a$

and *asso1*: $a \oplus b = \text{Some } ab \wedge b \oplus c = \text{Some } bc \implies ab \oplus c = a \oplus bc$

and *asso2*: $a \oplus b = \text{Some } ab \wedge \neg b \#\# c \implies \neg ab \#\# c$

and *sinv-inverse*: $\text{smult } p (\text{sinv } p) = \text{one}$

and *sone-neutral*: $\text{smult } \text{one } p = p$

and *sadd-comm*: $\text{sadd } p q = \text{sadd } q p$

and *smult-comm*: $\text{smult } p q = \text{smult } q p$

and *smult-distrib*: $\text{smult } p (\text{sadd } q r) = \text{sadd } (\text{smult } p q) (\text{smult } p r)$

and *smult-asso*: $\text{smult } (\text{smult } p q) r = \text{smult } p (\text{smult } q r)$

and *double-mult*: $p \odot (q \odot a) = (\text{smult } p q) \odot a$

and *plus-mult*: $\text{Some } a = b \oplus c \implies \text{Some } (p \odot a) = (p \odot b) \oplus (p \odot c)$

and *distrib-mult*: $\text{Some } ((\text{sadd } p q) \odot x) = p \odot x \oplus q \odot x$

and *one-neutral*: $\text{one} \odot a = a$

and *valid-mono*: $\text{valid } a \wedge a \succeq b \implies \text{valid } b$

begin

The validity of assertions corresponds to Figure 3 of the paper [5].

fun *sat* :: 'a \Rightarrow ('d \Rightarrow 'c) \Rightarrow ('d, 'c, 'a) *interp* \Rightarrow ('a, 'b, 'c, 'd) *assertion* \Rightarrow bool

(\Leftarrow , \neg , $- \models -$) [51, 65, 68, 66] 50) **where**

$\sigma, s, \Delta \models \text{Mult } p A \iff (\exists a. \sigma = p \odot a \wedge a, s, \Delta \models A)$

$|\sigma, s, \Delta \models \text{Star } A B \iff (\exists a b. \text{Some } \sigma = a \oplus b \wedge a, s, \Delta \models A \wedge b, s, \Delta \models B)$

$|\sigma, s, \Delta \models \text{Wand } A B \iff (\forall a \sigma'. a, s, \Delta \models A \wedge \text{Some } \sigma' = \sigma \oplus a \longrightarrow \sigma', s, \Delta \models B)$

$|\sigma, s, \Delta \models \text{Sem } b \iff b s \sigma$

$|\sigma, s, \Delta \models \text{Imp } A B \iff (\sigma, s, \Delta \models A \longrightarrow \sigma, s, \Delta \models B)$

$|\sigma, s, \Delta \models \text{Or } A B \iff (\sigma, s, \Delta \models A \vee \sigma, s, \Delta \models B)$

$|\sigma, s, \Delta \models \text{And } A B \iff (\sigma, s, \Delta \models A \wedge \sigma, s, \Delta \models B)$

$|\sigma, s, \Delta \models \text{Exists } x A \iff (\exists v. \sigma, s(x := v), \Delta \models A)$

$|\sigma, s, \Delta \models \text{Forall } x A \iff (\forall v. \sigma, s(x := v), \Delta \models A)$

$|\sigma, s, \Delta \models \text{Pred} \iff (\sigma \in \Delta s)$

$|\sigma, s, \Delta \models \text{Bounded } A \iff (\text{valid } \sigma \longrightarrow \sigma, s, \Delta \models A)$

$|\sigma, s, \Delta \models \text{Wildcard } A \iff (\exists a p. \sigma = p \odot a \wedge a, s, \Delta \models A)$

definition *intuitionistic* :: ('d \Rightarrow 'c) \Rightarrow ('d, 'c, 'a) *interp* \Rightarrow ('a, 'b, 'c, 'd) *assertion*

\Rightarrow bool **where**

intuitionistic $s \Delta A \iff (\forall a b. a \succeq b \wedge b, s, \Delta \models A \longrightarrow a, s, \Delta \models A)$

definition *entails* :: ('a, 'b, 'c, 'd) assertion \Rightarrow ('d, 'c, 'a) interp \Rightarrow ('a, 'b, 'c, 'd) assertion \Rightarrow bool ($\langle -, - \vdash - \rangle$ [63, 66, 68] 52) **where**
 $A, \Delta \vdash B \longleftrightarrow (\forall \sigma s. \sigma, s, \Delta \models A \longrightarrow \sigma, s, \Delta \models B)$

definition *equivalent* :: ('a, 'b, 'c, 'd) assertion \Rightarrow ('d, 'c, 'a) interp \Rightarrow ('a, 'b, 'c, 'd) assertion \Rightarrow bool ($\langle -, - \equiv - \rangle$ [63, 66, 68] 52) **where**
 $A, \Delta \equiv B \longleftrightarrow (A, \Delta \vdash B \wedge B, \Delta \vdash A)$

definition *pure* :: ('a, 'b, 'c, 'd) assertion \Rightarrow bool **where**
 $pure A \longleftrightarrow (\forall \sigma \sigma' s \Delta \Delta'. \sigma, s, \Delta \models A \longleftrightarrow \sigma', s, \Delta' \models A)$

1.2 Useful lemmas

lemma *sat-forall*:
assumes $\bigwedge v. \sigma, s(x := v), \Delta \models A$
shows $\sigma, s, \Delta \models \text{Forall } x A$
by (*simp add: assms*)

lemma *intuitionisticI*:
assumes $\bigwedge a b. a \succeq b \wedge b, s, \Delta \models A \Longrightarrow a, s, \Delta \models A$
shows *intuitionistic* $s \Delta A$
by (*meson assms intuitionistic-def*)

lemma *can-divide*:
assumes $p \odot a = p \odot b$
shows $a = b$
by (*metis assms double-mult logic.one-neutral logic-axioms sinv-inverse smult-comm*)

lemma *unique-inv*:
 $a = p \odot b \longleftrightarrow b = (\text{sinv } p) \odot a$
by (*metis double-mult logic.can-divide logic-axioms sinv-inverse sone-neutral*)

lemma *entailsI*:
assumes $\bigwedge \sigma s. \sigma, s, \Delta \models A \Longrightarrow \sigma, s, \Delta \models B$
shows $A, \Delta \vdash B$
by (*simp add: assms entails-def*)

lemma *equivalentI*:
assumes $\bigwedge \sigma s. \sigma, s, \Delta \models A \Longrightarrow \sigma, s, \Delta \models B$
and $\bigwedge \sigma s. \sigma, s, \Delta \models B \Longrightarrow \sigma, s, \Delta \models A$
shows $A, \Delta \equiv B$
by (*simp add: assms(1) assms(2) entailsI equivalent-def*)

lemma *compatible-imp*:
assumes $a \#\# b$
shows $(p \odot a) \#\# (p \odot b)$
by (*metis assms compatible-def option.distinct(1) option.exhaust plus-mult*)

lemma compatible-iff:

$a \#\# b \longleftrightarrow (p \odot a) \#\# (p \odot b)$
by (*metis compatible-imp unique-inv*)

lemma sat-wand:

assumes $\bigwedge a \sigma'. a, s, \Delta \models A \wedge \text{Some } \sigma' = \sigma \oplus a \implies \sigma', s, \Delta \models B$
shows $\sigma, s, \Delta \models \text{Wand } A B$
using *assms* **by** *auto*

lemma sat-imp:

assumes $\sigma, s, \Delta \models A \implies \sigma, s, \Delta \models B$
shows $\sigma, s, \Delta \models \text{Imp } A B$
using *assms* **by** *auto*

lemma sat-mult:

assumes $\bigwedge a. \sigma = p \odot a \implies a, s, \Delta \models A$
shows $\sigma, s, \Delta \models \text{Mult } p A$
by (*metis assms logic.sat.simps(1) logic-axioms unique-inv*)

lemma larger-same:

$a \succeq b \longleftrightarrow p \odot a \succeq p \odot b$
proof –
have $\bigwedge a b p. a \succeq b \implies p \odot a \succeq p \odot b$
by (*meson larger-def plus-mult*)
then show *?thesis*
by (*metis unique-inv*)
qed

lemma asso3:

assumes $\neg a \#\# b$
and $b \oplus c = \text{Some } bc$
shows $\neg a \#\# bc$
by (*metis (full-types) assms(1) assms(2) asso2 commutative compatible-def*)

lemma compatible-smaller:

assumes $a \succeq b$
and $x \#\# a$
shows $x \#\# b$
by (*metis assms(1) assms(2) asso3 larger-def*)

lemma compatible-multiples:

assumes $p \odot a \#\# q \odot b$
shows $a \#\# b$
by (*metis (no-types, opaque-lifting) assms commutative compatible-def compatible-iff compatible-smaller distrib-mult larger-def one-neutral*)

lemma move-sum:

assumes $\text{Some } a = a1 \oplus a2$
and $\text{Some } b = b1 \oplus b2$

and *Some* $x = a \oplus b$
and *Some* $x1 = a1 \oplus b1$
and *Some* $x2 = a2 \oplus b2$
shows *Some* $x = x1 \oplus x2$
proof –
obtain *ab1* **where** *Some* $ab1 = a \oplus b1$
by (*metis* *assms*(2) *assms*(3) *asso3* *compatible-def* *not-Some-eq*)
then have *Some* $ab1 = x1 \oplus a2$
by (*metis* *assms*(1) *assms*(4) *asso1* *commutative*)
then show *?thesis*
by (*metis* \langle *Some* $ab1 = a \oplus b1 \rangle$ *assms*(2) *assms*(3) *assms*(5) *asso1*)
qed

lemma *sum-both-larger*:
assumes *Some* $x' = a' \oplus b'$
and *Some* $x = a \oplus b$
and $a' \succeq a$
and $b' \succeq b$
shows $x' \succeq x$
proof –
obtain *ra* *rb* **where** *Some* $a' = a \oplus ra$ *Some* $b' = b \oplus rb$
using *assms*(3) *assms*(4) *larger-def* **by** *auto*
then obtain *r* **where** *Some* $r = ra \oplus rb$
by (*metis* *assms*(1) *asso3* *commutative* *compatible-def* *option.collapse*)
then have *Some* $x' = x \oplus r$
by (*meson* \langle *Some* $a' = a \oplus ra \rangle$ \langle *Some* $b' = b \oplus rb \rangle$ *assms*(1) *assms*(2)
move-sum)
then show *?thesis*
using *larger-def* **by** *blast*
qed

lemma *larger-first-sum*:
assumes *Some* $y = a \oplus b$
and $x \succeq y$
shows $\exists a'. \text{Some } x = a' \oplus b \wedge a' \succeq a$
proof –
obtain *r* **where** *Some* $x = y \oplus r$
using *assms*(2) *larger-def* **by** *auto*
then obtain *a'* **where** *Some* $a' = a \oplus r$
by (*metis* *assms*(1) *asso2* *commutative* *compatible-def* *option.collapse*)
then show *?thesis*
by (*metis* \langle *Some* $x = y \oplus r \rangle$ *assms*(1) *asso1* *commutative* *larger-def*)
qed

lemma *larger-implies-compatible*:
assumes $x \succeq y$
shows $x \#\# y$
by (*metis* *assms* *compatible-def* *compatible-smaller* *distrib-mult* *one-neutral* *option.distinct*(1))

2 Frame rule

This section corresponds to Section 2.5 of the paper [5].

definition *safe* :: ('a × ('d ⇒ 'c)) command ⇒ ('a × ('d ⇒ 'c)) ⇒ bool **where**
safe c σ ⇔ (σ, None) ∉ c

definition *safety-monotonicity* :: ('a × ('d ⇒ 'c)) command ⇒ bool **where**
safety-monotonicity c ⇔ (∀ σ σ' s. valid σ' ∧ σ' ≥ σ ∧ *safe* c (σ, s) → *safe* c (σ', s))

definition *frame-property* :: ('a × ('d ⇒ 'c)) command ⇒ bool **where**
frame-property c ⇔ (∀ σ σ0 r σ' s s'. valid σ ∧ valid σ' ∧ *safe* c (σ0, s) ∧ Some σ = σ0 ⊕ r ∧ ((σ, s), Some (σ', s')) ∈ c → (∃ σ0'. Some σ' = σ0' ⊕ r ∧ ((σ0', s), Some (σ0', s')) ∈ c))

definition *valid-hoare-triple* :: ('a, 'b, 'c, 'd) assertion ⇒ ('a × ('d ⇒ 'c)) command ⇒ ('a, 'b, 'c, 'd) assertion ⇒ ('d, 'c, 'a) interp ⇒ bool **where**
valid-hoare-triple P c Q Δ ⇔ (∀ σ s. valid σ ∧ σ, s, Δ ⊨ P → *safe* c (σ, s) ∧ (∀ σ' s'. ((σ, s), Some (σ', s')) ∈ c → σ', s', Δ ⊨ Q))

lemma *valid-hoare-tripleI*:

assumes ∧σ s. valid σ ∧ σ, s, Δ ⊨ P ⇒ *safe* c (σ, s)
and ∧σ s σ' s'. valid σ ∧ σ, s, Δ ⊨ P ⇒ ((σ, s), Some (σ', s')) ∈ c ⇒ σ', s', Δ ⊨ Q
shows *valid-hoare-triple* P c Q Δ
using *assms(1) assms(2) valid-hoare-triple-def by blast*

definition *valid-command* :: ('a × ('d ⇒ 'c)) command ⇒ bool **where**
valid-command c ⇔ (∀ a b sa sb. ((a, sa), Some (b, sb)) ∈ c ∧ valid a → valid b)

definition *modified* :: ('a × ('d ⇒ 'c)) command ⇒ 'd set **where**
modified c = { x | x. ∃ σ s σ' s'. ((σ, s), Some (σ', s')) ∈ c ∧ s x ≠ s' x }

definition *equal-outside* :: ('d ⇒ 'c) ⇒ ('d ⇒ 'c) ⇒ 'd set ⇒ bool **where**
equal-outside s s' S ⇔ (∀ x. x ∉ S → s x = s' x)

definition *not-in-fv* :: ('a, 'b, 'c, 'd) assertion ⇒ 'd set ⇒ bool **where**
not-in-fv A S ⇔ (∀ σ s Δ s'. *equal-outside* s s' S → (σ, s, Δ ⊨ A ⇔ σ, s', Δ ⊨ A))

lemma *not-in-fv-mod*:

assumes *not-in-fv* A (*modified* c)
and ((σ, s), Some (σ', s')) ∈ c
shows x, s, Δ ⊨ A ⇔ x, s', Δ ⊨ A

```

proof –
  have  $\bigwedge x. x \notin (\text{modified } c) \implies s\ x = s'\ x$ 
  proof –
    fix  $x$  assume  $x \notin (\text{modified } c)$ 
    then show  $s\ x = s'\ x$ 
      by (metis (mono-tags, lifting) CollectI assms(2) modified-def)
    qed
  then have equal-outside  $s\ s'$  (modified  $c$ )
    by (simp add: equal-outside-def)
  then show ?thesis
    using assms(1) not-in-fv-def by blast
qed

```

This theorem corresponds to Theorem 2 of the paper [5].

```

theorem frame-rule:
  assumes valid-command  $c$ 
    and safety-monotonicity  $c$ 
    and frame-property  $c$ 
    and valid-hoare-triple  $P\ c\ Q\ \Delta$ 
    and not-in-fv  $R$  (modified  $c$ )
  shows valid-hoare-triple (Star  $P\ R$ )  $c$  (Star  $Q\ R$ )  $\Delta$ 
proof (rule valid-hoare-tripleI)
  fix  $\sigma\ s$  assume asm0: valid  $\sigma \wedge \sigma, s, \Delta \models \text{Star } P\ R$ 
  then obtain  $p\ r$  where Some  $\sigma = p \oplus r$   $p, s, \Delta \models P\ r, s, \Delta \models R$ 
    by auto
  then have safe  $c$  ( $p, s$ )
    by (metis asm0 assms(4) larger-def logic.valid-mono logic-axioms valid-hoare-triple-def)
  then show safe  $c$  ( $\sigma, s$ )
    using  $\langle \text{Some } \sigma = p \oplus r \rangle$  assms(2) larger-def safety-monotonicity-def asm0 by
blast
  fix  $\sigma'\ s'$  assume asm1:  $((\sigma, s), \text{Some } (\sigma', s')) \in c$ 
  then obtain  $q$  where Some  $\sigma' = q \oplus r$   $((p, s), \text{Some } (q, s')) \in c$ 
    using  $\langle \text{Some } \sigma = p \oplus r \rangle$   $\langle \text{safe } c (p, s) \rangle$  asm0 assms(1) assms(3) frame-property-def
valid-command-def by blast
  moreover have  $r, s', \Delta \models R$ 
    by (meson  $\langle r, s, \Delta \models R \rangle$  assms(5) calculation(2) logic.not-in-fv-mod logic-axioms)
  ultimately show  $\sigma', s', \Delta \models \text{Star } Q\ R$ 
    by (meson  $\langle \text{Some } \sigma = p \oplus r \rangle$   $\langle p, s, \Delta \models P \rangle$   $\langle r, s, \Delta \models R \rangle$  asm0 assms(4)
larger-def sat.simps(2) valid-hoare-triple-def valid-mono)
qed

```

```

lemma hoare-triple-input:
  valid-hoare-triple  $P\ c\ Q\ \Delta \iff \text{valid-hoare-triple } (\text{Bounded } P)\ c\ Q\ \Delta$ 
  using sat.simps(11) valid-hoare-triple-def by blast

```

```

lemma hoare-triple-output:
  assumes valid-command  $c$ 

```

shows *valid-hoare-triple* $P\ c\ Q\ \Delta \longleftrightarrow \text{valid-hoare-triple } P\ c\ (\text{Bounded } Q)\ \Delta$
using *assms valid-command-def valid-hoare-triple-def* **by** *fastforce*

end

end

3 Distributivity and Factorisability

This section corresponds to Section 2.4 and Figure 4 of the paper [5].

theory *Distributivity*
imports *UnboundedLogic*
begin

context *logic*
begin

3.1 DotPos

lemma *DotPos*:
 $A, \Delta \vdash B \longleftrightarrow (\text{Mult } \pi\ A, \Delta \vdash \text{Mult } \pi\ B)$ (**is** $?A \longleftrightarrow ?B$)
proof
show $?A \implies ?B$
by (*metis (no-types, lifting) entails-def sat.simps(1)*)
show $?B \implies ?A$
using *can-divide entails-def sat.simps(1)*
by *metis*
qed

Only one direction holds with a wildcard

lemma *WildPos*:
 $A, \Delta \vdash B \implies (\text{Wildcard } A, \Delta \vdash \text{Wildcard } B)$
by (*metis (no-types, lifting) entails-def sat.simps(12)*)

3.2 DotDot

lemma *dot-mult1*:
 $\text{Mult } p\ (\text{Mult } q\ A), \Delta \vdash \text{Mult } (\text{smult } p\ q)\ A$
proof (*rule entailsI*)
fix $\sigma\ s$
assume $\sigma, s, \Delta \models \text{Mult } p\ (\text{Mult } q\ A)$
then show $\sigma, s, \Delta \models \text{Mult } (\text{smult } p\ q)\ A$
using *double-mult* **by** *auto*
qed

lemma *dot-mult2*:

$Mult (smult\ p\ q)\ A, \Delta \vdash Mult\ p\ (Mult\ q\ A)$
proof (*rule entailsI*)
fix $\sigma\ s\ \Delta$
assume $\sigma, s, \Delta \models Mult\ (smult\ p\ q)\ A$
then obtain a **where** $a, s, \Delta \models A\ \sigma = (smult\ p\ q) \odot a$
by *auto*
then have $q \odot a, s, \Delta \models Mult\ q\ A$ **by** *auto*
then show $\sigma, s, \Delta \models Mult\ p\ (Mult\ q\ A)$
by (*metis* $\langle \sigma = smult\ p\ q \odot a \rangle$ *double-mult sat.simps(1)*)
qed

lemma *DotDot*:
 $Mult\ p\ (Mult\ q\ A), \Delta \equiv Mult\ (smult\ p\ q)\ A$
by (*simp add: dot-mult1 dot-mult2 equivalent-def*)

lemma *can-factorize*:
 $\exists r. q = smult\ r\ p$
by (*metis sinv-inverse smult-asso smult-comm sone-neutral*)

lemma *WildDot*:
 $Wildcard\ (Mult\ p\ A), \Delta \equiv Wildcard\ A$
proof (*rule equivalentI*)
show $\bigwedge \sigma\ s. \sigma, s, \Delta \models Wildcard\ (Mult\ p\ A) \implies \sigma, s, \Delta \models Wildcard\ A$
using *double-mult by fastforce*
fix $\sigma\ s$
assume *asm0*: $\sigma, s, \Delta \models Wildcard\ A$
then obtain $q\ a$ **where** $\sigma = q \odot a, a, s, \Delta \models A$
using *sat.simps(12) by blast*
then obtain r **where** $q = smult\ r\ p$
using *can-factorize by blast*
then have $\sigma = r \odot (p \odot a)$
by (*simp add:* $\langle \sigma = q \odot a \rangle$ *double-mult*)
then show $\sigma, s, \Delta \models Wildcard\ (Mult\ p\ A)$
using $\langle a, s, \Delta \models A \rangle$ *sat.simps(1) sat.simps(12) by blast*
qed

lemma *DotWild*:
 $Mult\ p\ (Wildcard\ A), \Delta \equiv Wildcard\ A$
proof (*rule equivalentI*)
show $\bigwedge \sigma\ s. \sigma, s, \Delta \models Mult\ p\ (Wildcard\ A) \implies \sigma, s, \Delta \models Wildcard\ A$
using *double-mult by fastforce*
fix $\sigma\ s$
assume *asm0*: $\sigma, s, \Delta \models Wildcard\ A$
then obtain $q\ a$ **where** $\sigma = q \odot a, a, s, \Delta \models A$
by *force*
then obtain r **where** $q = smult\ p\ r$
using *can-factorize smult-comm by presburger*
then have $\sigma = p \odot (r \odot a)$
by (*simp add:* $\langle \sigma = q \odot a \rangle$ *double-mult*)

then show $\sigma, s, \Delta \models \text{Mult } p \text{ (Wildcard } A)$
using $\langle a, s, \Delta \models A \rangle$ **by** *auto*
qed

lemma *WildWild*:

Wildcard (Wildcard A), $\Delta \equiv \text{Wildcard } A$

proof (*rule equivalentI*)

show $\bigwedge \sigma s. \sigma, s, \Delta \models \text{Wildcard (Wildcard } A) \implies \sigma, s, \Delta \models \text{Wildcard } A$
using *double-mult* **by** *fastforce*

show $\bigwedge \sigma s. \sigma, s, \Delta \models \text{Wildcard } A \implies \sigma, s, \Delta \models \text{Wildcard (Wildcard } A)$
by (*metis one-neutral sat.simps(12)*)

qed

3.3 DotStar

lemma *dot-star1*:

Mult p (Star A B), $\Delta \vdash \text{Star (Mult p A) (Mult p B)}$

proof (*rule entailsI*)

fix $\sigma s \Delta$

assume $\sigma, s, \Delta \models \text{Mult } p \text{ (Star } A \ B)$

then obtain $a \ b \ x$ **where** $\sigma = p \odot x$ *Some* $x = a \oplus b$ $a, s, \Delta \models A$ $b, s, \Delta \models B$
by *auto*

then show $\sigma, s, \Delta \models \text{Star (Mult p A) (Mult p B)}$

using *plus-mult* **by** *auto*

qed

lemma *dot-star2*:

Star (Mult p A) (Mult p B), $\Delta \vdash \text{Mult } p \text{ (Star } A \ B)$

proof (*rule entailsI*)

fix $\sigma s \Delta$

assume $\sigma, s, \Delta \models \text{Star (Mult p A) (Mult p B)}$

then obtain $a \ b$ **where** *Some* $\sigma = (p \odot a) \oplus (p \odot b)$ $a, s, \Delta \models A$ $b, s, \Delta \models B$
by *auto*

then obtain x **where** *Some* $x = a \oplus b$

by (*metis plus-mult unique-inv*)

then have $\sigma = p \odot x$

by (*metis* $\langle \text{Some } \sigma = p \odot a \oplus p \odot b \rangle$ *option.sel plus-mult*)

then show $\sigma, s, \Delta \models \text{Mult } p \text{ (Star } A \ B)$

using $\langle \text{Some } x = a \oplus b \rangle \langle a, s, \Delta \models A \rangle \langle b, s, \Delta \models B \rangle$ **by** *auto*

qed

lemma *DotStar*:

Mult p (Star A B), $\Delta \equiv \text{Star (Mult p A) (Mult p B)}$

by (*simp add: dot-star1 dot-star2 equivalent-def*)

lemma *WildStar1*:

Wildcard (Star A B), $\Delta \vdash \text{Star (Wildcard } A) \text{ (Wildcard } B)$

proof (*rule entailsI*)

```

fix  $\sigma$   $s$  assume  $asm0: \sigma, s, \Delta \models \text{Wildcard } (\text{Star } A \ B)$ 
then obtain  $p \ ab \ a \ b$  where  $\sigma = p \odot ab$   $\text{Some } ab = a \oplus b$   $a, s, \Delta \models A$   $b, s, \Delta \models B$ 
by auto
then have  $\text{Some } \sigma = (p \odot a) \oplus (p \odot b)$ 
using plus-mult by blast
then show  $\sigma, s, \Delta \models \text{Star } (\text{Wildcard } A) (\text{Wildcard } B)$ 
using  $\langle a, s, \Delta \models A \rangle \langle b, s, \Delta \models B \rangle$  by auto
qed

```

3.4 DotWand

lemma *dot-wand1*:

```

 $\text{Mult } p \ (\text{Wand } A \ B), \Delta \vdash \text{Wand } (\text{Mult } p \ A) (\text{Mult } p \ B)$ 
proof (rule entailsI)
fix  $\sigma$   $s$   $\Delta$ 
assume  $\sigma, s, \Delta \models \text{Mult } p \ (\text{Wand } A \ B)$ 
then obtain  $x$  where  $\sigma = p \odot x$   $x, s, \Delta \models \text{Wand } A \ B$ 
by auto
show  $\sigma, s, \Delta \models \text{Wand } (\text{Mult } p \ A) (\text{Mult } p \ B)$ 
proof (rule sat-wand)
fix  $a$   $\sigma'$ 
assume  $a, s, \Delta \models \text{Mult } p \ A \wedge \text{Some } \sigma' = \sigma \oplus a$ 
then obtain  $aa$  where  $aa, s, \Delta \models A$   $a = p \odot aa$ 
by auto
then obtain  $b$  where  $\text{Some } b = x \oplus aa$ 
by (metis  $\langle \sigma = p \odot x \rangle \langle a, s, \Delta \models \text{Mult } p \ A \wedge \text{Some } \sigma' = \sigma \oplus a \rangle$  compatible-def
compatible-iff option.exhaust-sel)
then have  $b, s, \Delta \models B$ 
using  $\langle aa, s, \Delta \models A \rangle \langle x, s, \Delta \models \text{Wand } A \ B \rangle$  by auto
then show  $\sigma', s, \Delta \models \text{Mult } p \ B$ 
by (metis  $\langle \text{Some } b = x \oplus aa \rangle \langle \sigma = p \odot x \rangle \langle a = p \odot aa \rangle \langle a, s, \Delta \models \text{Mult } p \ A \wedge \text{Some } \sigma' = \sigma \oplus a \rangle$  can-divide option.inject plus-mult sat-mult)
qed
qed

```

lemma *dot-wand2*:

```

 $\text{Wand } (\text{Mult } p \ A) (\text{Mult } p \ B), \Delta \vdash \text{Mult } p \ (\text{Wand } A \ B)$ 
proof (rule entailsI)
fix  $\sigma$   $s$   $\Delta$ 
assume  $asm: \sigma, s, \Delta \models \text{Wand } (\text{Mult } p \ A) (\text{Mult } p \ B)$ 
show  $\sigma, s, \Delta \models \text{Mult } p \ (\text{Wand } A \ B)$ 
proof (rule sat-mult)
fix  $a$  assume  $\sigma = p \odot a$ 
show  $a, s, \Delta \models \text{Wand } A \ B$ 
proof (rule sat-wand)
fix  $aa$   $\sigma'$ 
assume  $aa, s, \Delta \models A \wedge \text{Some } \sigma' = a \oplus aa$ 
then have  $p \odot aa, s, \Delta \models \text{Mult } p \ A$  by auto

```

```

then have Some (p ⊙ σ') = σ ⊕ p ⊙ aa
  by (simp add: <σ = p ⊙ a> <aa, s, Δ ⊢ A ∧ Some σ' = a ⊕ aa> plus-mult)
then have p ⊙ σ', s, Δ ⊢ Mult p B
  using <p ⊙ aa, s, Δ ⊢ Mult p A> asm by force
then show σ', s, Δ ⊢ B
  by (metis can-divide sat.simps(1))
qed
qed
qed

```

lemma *DotWand*:

```

Mult p (Wand A B), Δ ≡ Wand (Mult p A) (Mult p B)
by (simp add: dot-wand1 dot-wand2 equivalent-def)

```

3.5 DotOr

lemma *dot-or1*:

```

Mult p (Or A B), Δ ⊢ Or (Mult p A) (Mult p B)
proof (rule entailsI)
  fix σ s Δ
  assume σ, s, Δ ⊢ Mult p (Or A B)
  then obtain x where σ = p ⊙ x x, s, Δ ⊢ A ∨ x, s, Δ ⊢ B
    by auto
  then show σ, s, Δ ⊢ Or (Mult p A) (Mult p B)
  proof (cases x, s, Δ ⊢ A)
    case True
      then show ?thesis
      using <σ = p ⊙ x> by auto
    next
      case False
      then show ?thesis
      using <σ = p ⊙ x> <x, s, Δ ⊢ A ∨ x, s, Δ ⊢ B> by auto
  qed
qed

```

lemma *dot-or2*:

```

Or (Mult p A) (Mult p B), Δ ⊢ Mult p (Or A B)
proof (rule entailsI)
  fix σ s Δ
  assume σ, s, Δ ⊢ Or (Mult p A) (Mult p B)
  then show σ, s, Δ ⊢ Mult p (Or A B)
  proof (cases σ, s, Δ ⊢ Mult p A)
    case True
      then show ?thesis by auto
    next
      case False
      then show ?thesis
      using <σ, s, Δ ⊢ Or (Mult p A) (Mult p B)> by auto
  qed
qed

```

qed

lemma *DotOr*:

$Mult\ p\ (Or\ A\ B),\ \Delta \equiv Or\ (Mult\ p\ A)\ (Mult\ p\ B)$
by (*simp add: dot-or1 dot-or2 equivalent-def*)

lemma *WildOr*:

$Wildcard\ (Or\ A\ B),\ \Delta \equiv Or\ (Wildcard\ A)\ (Wildcard\ B)$

proof (*rule equivalentI*)

show $\bigwedge\sigma\ s.\ \sigma,\ s,\ \Delta \models Wildcard\ (Or\ A\ B) \implies \sigma,\ s,\ \Delta \models Or\ (Wildcard\ A)$
 $(Wildcard\ B)$

by *auto*

show $\bigwedge\sigma\ s.\ \sigma,\ s,\ \Delta \models Or\ (Wildcard\ A)\ (Wildcard\ B) \implies \sigma,\ s,\ \Delta \models Wildcard$
 $(Or\ A\ B)$

by *auto*

qed

3.6 DotAnd

lemma *dot-and1*:

$Mult\ p\ (And\ A\ B),\ \Delta \vdash And\ (Mult\ p\ A)\ (Mult\ p\ B)$

proof (*rule entailsI*)

fix $\sigma\ s\ \Delta$

assume $\sigma,\ s,\ \Delta \models Mult\ p\ (And\ A\ B)$

then obtain x **where** $\sigma = p \odot x\ x,\ s,\ \Delta \models A\ x,\ s,\ \Delta \models B$

by *auto*

then show $\sigma,\ s,\ \Delta \models And\ (Mult\ p\ A)\ (Mult\ p\ B)$

by *auto*

qed

lemma *dot-and2*:

$And\ (Mult\ p\ A)\ (Mult\ p\ B),\ \Delta \vdash Mult\ p\ (And\ A\ B)$

proof (*rule entailsI*)

fix $\sigma\ s\ \Delta$

assume $\sigma,\ s,\ \Delta \models And\ (Mult\ p\ A)\ (Mult\ p\ B)$

then show $\sigma,\ s,\ \Delta \models Mult\ p\ (And\ A\ B)$

using *logic.can-divide logic-axioms* **by** *fastforce*

qed

lemma *DotAnd*:

$And\ (Mult\ p\ A)\ (Mult\ p\ B),\ \Delta \equiv Mult\ p\ (And\ A\ B)$

by (*simp add: dot-and1 dot-and2 equivalent-def*)

lemma *WildAnd*:

$Wildcard\ (And\ A\ B),\ \Delta \vdash And\ (Wildcard\ A)\ (Wildcard\ B)$

using *entails-def* **by** *fastforce*

3.7 DotImp

lemma *dot-imp1*:

```

  Imp (Mult p A) (Mult p B), Δ ⊢ Mult p (Imp A B)
proof (rule entailsI)
  fix σ s Δ
  assume σ, s, Δ ⊨ Imp (Mult p A) (Mult p B)
  then show σ, s, Δ ⊨ Mult p (Imp A B)
    using sat-mult by force
qed

```

```

lemma dot-imp2:
  Mult p (Imp A B), Δ ⊢ Imp (Mult p A) (Mult p B)
proof (rule entailsI)
  fix σ s Δ
  assume σ, s, Δ ⊨ Mult p (Imp A B)
  then show σ, s, Δ ⊨ Imp (Mult p A) (Mult p B)
    using can-divide by auto
qed

```

```

lemma DotImp:
  Mult p (Imp A B), Δ ≡ Imp (Mult p A) (Mult p B)
  by (simp add: dot-imp1 dot-imp2 equivalent-def)

```

3.8 DotPure

```

lemma pure-mult1:
  assumes pure A
  shows Mult p A, Δ ⊢ A
  using assms entails-def logic.pure-def logic-axioms by fastforce

```

```

lemma pure-mult2:
  assumes pure A
  shows A, Δ ⊢ Mult p A
  using assms entailsI pure-def sat-mult
  by metis

```

```

lemma DotPure:
  assumes pure A
  shows Mult p A, Δ ≡ A
  by (simp add: assms equivalent-def pure-mult1 pure-mult2)

```

```

lemma WildPure:
  assumes pure A
  shows Wildcard A, Δ ≡ A
proof (rule equivalentI)
  show ∧σ s. σ, s, Δ ⊨ Wildcard A ⇒ σ, s, Δ ⊨ A
    using assms pure-def sat.simps(12) by blast
  show ∧σ s. σ, s, Δ ⊨ A ⇒ σ, s, Δ ⊨ Wildcard A
    by (metis one-neutral sat.simps(12))
qed

```

3.9 DotFull

lemma *mult-one-same1*:
 $Mult\ one\ A, \Delta \vdash A$
 by (*simp add: entails-def one-neutral*)

lemma *mult-one-same2*:
 $A, \Delta \vdash Mult\ one\ A$
 by (*simp add: entailsI one-neutral*)

lemma *DotFull*:
 $Mult\ one\ A, \Delta \equiv A$
 using *equivalent-def mult-one-same1 mult-one-same2* **by** *blast*

3.10 DotExists

lemma *dot-exists1*:
 $Mult\ p\ (Exists\ x\ A), \Delta \vdash Exists\ x\ (Mult\ p\ A)$
proof (*rule entailsI*)
 fix $\sigma\ s\ \Delta$
 assume $\sigma, s, \Delta \models Mult\ p\ (Exists\ x\ A)$
 then show $\sigma, s, \Delta \models Exists\ x\ (Mult\ p\ A)$
 by *auto*
qed

lemma *dot-exists2*:
 $Exists\ x\ (Mult\ p\ A), \Delta \vdash Mult\ p\ (Exists\ x\ A)$
proof (*rule entailsI*)
 fix $\sigma\ s\ \Delta$
 assume $\sigma, s, \Delta \models Exists\ x\ (Mult\ p\ A)$
 then show $\sigma, s, \Delta \models Mult\ p\ (Exists\ x\ A)$ **by** *auto*
qed

lemma *DotExists*:
 $Mult\ p\ (Exists\ x\ A), \Delta \equiv Exists\ x\ (Mult\ p\ A)$
 by (*simp add: dot-exists1 dot-exists2 equivalent-def*)

lemma *WildExists*:
 $Wildcard\ (Exists\ x\ A), \Delta \equiv Exists\ x\ (Wildcard\ A)$
proof (*rule equivalentI*)
 show $\bigwedge \sigma\ s. \sigma, s, \Delta \models Wildcard\ (Exists\ x\ A) \implies \sigma, s, \Delta \models Exists\ x\ (Wildcard\ A)$
 by *auto*
 show $\bigwedge \sigma\ s. \sigma, s, \Delta \models Exists\ x\ (Wildcard\ A) \implies \sigma, s, \Delta \models Wildcard\ (Exists\ x\ A)$
 by *auto*
qed

3.11 DotForall

lemma *dot-forall1*:

$Mult\ p\ (Forall\ x\ A), \Delta \vdash Forall\ x\ (Mult\ p\ A)$

proof (*rule entailsI*)

fix $\sigma\ s\ \Delta$

assume $\sigma, s, \Delta \models Mult\ p\ (Forall\ x\ A)$

then show $\sigma, s, \Delta \models Forall\ x\ (Mult\ p\ A)$

by *auto*

qed

lemma *dot-forall2*:

$Forall\ x\ (Mult\ p\ A), \Delta \vdash Mult\ p\ (Forall\ x\ A)$

proof (*rule entailsI*)

fix $\sigma\ s\ \Delta$

assume $\sigma, s, \Delta \models Forall\ x\ (Mult\ p\ A)$

obtain a **where** $\sigma = p \odot a$

using *sat.simps(1) sat-mult by blast*

have $a, s, \Delta \models Forall\ x\ A$

proof (*rule sat-forall*)

fix v **show** $a, s(x := v), \Delta \models A$

using $\langle \sigma = p \odot a \rangle \langle \sigma, s, \Delta \models Forall\ x\ (Mult\ p\ A) \rangle$ *can-divide by auto*

qed

then show $\sigma, s, \Delta \models Mult\ p\ (Forall\ x\ A)$

using $\langle \sigma = p \odot a \rangle$ **by** *auto*

qed

lemma *DotForall*:

$Mult\ p\ (Forall\ x\ A), \Delta \equiv Forall\ x\ (Mult\ p\ A)$

by (*simp add: dot-forall1 dot-forall2 equivalent-def*)

lemma *WildForall*:

Wildcard $(Forall\ x\ A), \Delta \vdash Forall\ x\ (Wildcard\ A)$

by (*metis (no-types, lifting) entailsI sat.simps(12) sat.simps(9)*)

3.12 Split

lemma *split*:

$Mult\ (sadd\ a\ b)\ A, \Delta \vdash Star\ (Mult\ a\ A)\ (Mult\ b\ A)$

proof (*rule entailsI*)

fix $\sigma\ s$

assume $\sigma, s, \Delta \models Mult\ (sadd\ a\ b)\ A$

then show $\sigma, s, \Delta \models Star\ (Mult\ a\ A)\ (Mult\ b\ A)$

using *distrib-mult by fastforce*

qed

end

end

4 Combinability

This section corresponds to Section 3 of the paper [5].

```
theory Combinability
  imports UnboundedLogic
begin
```

```
context logic
begin
```

The definition of combinable assertions corresponds to Definition 4 of the paper [5].

```
definition combinable :: ('d, 'c, 'a) interp  $\Rightarrow$  ('a, 'b, 'c, 'd) assertion  $\Rightarrow$  bool
where
```

```
  combinable  $\Delta$  A  $\longleftrightarrow$  ( $\forall p q.$  Star (Mult p A) (Mult q A),  $\Delta \vdash$  Mult (sadd p q) A)
```

```
lemma combinable-instantiate:
```

```
  assumes combinable  $\Delta$  A
    and a, s,  $\Delta \models$  A
    and b, s,  $\Delta \models$  A
    and Some x = p  $\odot$  a  $\oplus$  q  $\odot$  b
  shows x, s,  $\Delta \models$  Mult (sadd p q) A
  by (meson assms(1) assms(2) assms(3) assms(4) combinable-def entails-def
logic.sat.simps(2) logic-axioms sat.simps(1))
```

```
lemma combinable-instantiate-one:
```

```
  assumes combinable  $\Delta$  A
    and a, s,  $\Delta \models$  A
    and b, s,  $\Delta \models$  A
    and Some x = p  $\odot$  a  $\oplus$  q  $\odot$  b
    and sadd p q = one
  shows x, s,  $\Delta \models$  A
  using assms(1) assms(2) assms(3) assms(4) assms(5) combinable-instantiate
one-neutral by fastforce
```

```
lemma combinableI-old:
```

```
  assumes  $\bigwedge a b p q x \sigma s.$  a, s,  $\Delta \models$  A  $\wedge$  b, s,  $\Delta \models$  A  $\wedge$  Some  $\sigma$  = p  $\odot$  a  $\oplus$  q  $\odot$ 
b  $\wedge$   $\sigma$  = (sadd p q)  $\odot$  x  $\Longrightarrow$  x, s,  $\Delta \models$  A
  shows combinable  $\Delta$  A
```

```
proof -
```

```
  have  $\bigwedge p q.$  Star (Mult p A) (Mult q A),  $\Delta \vdash$  Mult (sadd p q) A
```

```
  proof (rule entailsI)
```

```
    fix p q  $\sigma$  s
```

```
    assume  $\sigma$ , s,  $\Delta \models$  Star (Mult p A) (Mult q A)
```

```
    then obtain a b where a, s,  $\Delta \models$  A  $\wedge$  b, s,  $\Delta \models$  A  $\wedge$  Some  $\sigma$  = p  $\odot$  a  $\oplus$  q
 $\odot$  b
```

```
      by auto
```

```
    moreover obtain x where  $\sigma$  = (sadd p q)  $\odot$  x
```

using *unique-inv* **by** *auto*
ultimately have $x, s, \Delta \models A$ **using** *assms*
by *blast*
then show $\sigma, s, \Delta \models \text{Mult } (\text{sadd } p \ q) \ A$
using $\langle \sigma = \text{sadd } p \ q \odot x \rangle$ **by** *fastforce*
qed
then show *?thesis*
by (*simp add: combinable-def*)
qed

lemma *combinableI*:

assumes $\bigwedge a \ b \ p \ q \ x \ \sigma \ s. a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one} \implies x, s, \Delta \models A$
shows *combinable* $\Delta \ A$
proof (*rule combinableI-old*)
fix $a \ b \ p \ q \ x \ \sigma \ s$
assume $a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x$
let $?p = \text{smult } (\text{sinv } (\text{sadd } p \ q)) \ p$
let $?q = \text{smult } (\text{sinv } (\text{sadd } p \ q)) \ q$
have $\text{Some } x = ?p \odot a \oplus ?q \odot b$
proof –
have $\text{Some } ((\text{smult } (\text{sinv } (\text{sadd } p \ q)) \ (\text{sadd } p \ q)) \odot x) = ?p \odot a \oplus ?q \odot b$
by (*metis* $\langle a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$ *double-mult logic.plus-mult logic-axioms*)
then show *?thesis*
by (*simp add: one-neutral sinv-inverse smult-comm*)
qed
moreover have $\text{sadd } ?p \ ?q = \text{one}$
by (*metis logic.smult-comm logic-axioms sinv-inverse smult-distrib*)
ultimately show $x, s, \Delta \models A$
using $\langle a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$ *assms* **by** *blast*
qed

lemma *combinable-wand*:

assumes *combinable* $\Delta \ B$
shows *combinable* $\Delta \ (\text{Wand } A \ B)$
proof (*rule combinableI-old*)
fix $a \ b \ p \ q \ x \ \sigma \ s$
assume $a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x$
show $x, s, \Delta \models \text{Wand } A \ B$
proof (*rule sat-wand*)
fix $aa \ \sigma'$
assume $aa, s, \Delta \models A \wedge \text{Some } \sigma' = x \oplus aa$
then have $\text{Some } ((\text{sadd } p \ q) \odot \sigma') = \sigma \oplus ((\text{sadd } p \ q) \odot aa)$

by (simp add: $\langle a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p$
 $\odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$ plus-mult)

moreover have $\text{Some } ((\text{sadd } p \ q) \odot aa) = p \odot aa \oplus q \odot aa$

by (simp add: distrib-mult)

moreover have $a \ \#\# \ aa$

proof –

have $p \odot a \ \#\# \ (\text{sadd } p \ q) \odot aa$

by (metis $\langle a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p \odot$
 $a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$ asso2 calculation(1) commutative compatible-def
 option.discI)

then show ?thesis

using compatible-multiples by blast

qed

then obtain aaa where $\text{Some } aaa = a \oplus aa$

using compatible-def by auto

moreover have $b \ \#\# \ aa$

proof –

have $q \odot b \ \#\# \ (\text{sadd } p \ q) \odot aa$

by (metis $\langle a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p \odot$
 $a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$ asso2 calculation(1) compatible-def option.discI)

then show ?thesis

using compatible-multiples by blast

qed

then obtain baa where $\text{Some } baa = b \oplus aa$

using compatible-def by auto

ultimately have $\text{Some } (\text{mult } (\text{sadd } p \ q) \ \sigma') = p \odot aaa \oplus q \odot baa$

proof –

obtain $a1$ where $\text{Some } a1 = \sigma \oplus (p \odot aa)$

by (metis $\langle \text{Some } (\text{sadd } p \ q \odot \sigma') = \sigma \oplus \text{sadd } p \ q \odot aa \rangle$ compatible-multiples
 option.exhaust-sel pre-logic.compatible-def unique-inv)

then obtain $a2$ where $\text{Some } a2 = p \odot a \oplus (p \odot aa)$

by (meson $\langle \wedge \text{thesis. } (\wedge aaa. \text{Some } aaa = a \oplus aa \implies \text{thesis}) \implies \text{thesis} \rangle$
 plus-mult)

then have $\text{Some } a1 = a2 \oplus q \odot b$

proof –

obtain bc where $q \odot b \oplus p \odot aa = \text{Some } bc$

by (metis $\langle b \ \#\# \ aa \rangle$ compatible-iff compatible-multiples one-neutral
 option.exhaust-sel pre-logic.compatible-def)

then have $\sigma \oplus p \odot aa = p \odot a \oplus bc$

using asso1[of $p \odot a \ q \odot b \ \sigma \ p \odot aa \ bc$]

by (metis $\langle a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p$
 $\odot a \oplus q \odot b \wedge \sigma = \text{sadd } p \ q \odot x \rangle$)

then show ?thesis

by (metis $\langle \text{Some } a1 = \sigma \oplus p \odot aa \rangle \langle \text{Some } a2 = p \odot a \oplus p \odot aa \rangle \langle q \odot b$
 $\oplus p \odot aa = \text{Some } bc \rangle$ asso1 commutative)

qed

moreover have $a2 = p \odot aaa$

by (metis $\langle \text{Some } a2 = p \odot a \oplus p \odot aa \rangle \langle \text{Some } aaa = a \oplus aa \rangle$ option.inject
 plus-mult)

moreover have $\text{Some } (q \odot \text{baa}) = q \odot b \oplus q \odot \text{aa}$
by (*simp add: <Some baa = b \oplus aa> plus-mult*)
ultimately show *?thesis*
by (*metis <Some (sadd p q \odot σ') = $\sigma \oplus \text{sadd p q \odot aa> <Some (sadd p q \odot aa) = p \odot aa \oplus q \odot aa> <Some a1 = $\sigma \oplus p \odot aa> asso1$$*)
qed
moreover have $\text{aaa}, s, \Delta \models B \wedge \text{baa}, s, \Delta \models B$
using $\langle \text{Some aaa} = a \oplus \text{aa} \rangle \langle \text{Some baa} = b \oplus \text{aa} \rangle \langle a, s, \Delta \models \text{Wand } A \ B \wedge b, s, \Delta \models \text{Wand } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd p q \odot x} \rangle \langle \text{aa}, s, \Delta \models A \wedge \text{Some } \sigma' = x \oplus \text{aa} \rangle$ **by** *auto*
ultimately have $\text{mult } (\text{sadd p q}) \ \sigma', s, \Delta \models \text{Mult } (\text{sadd p q}) \ B$
by (*meson assms logic.combinable-def logic.entails-def logic-axioms sat.simps(1) sat.simps(2)*)
then show $\sigma', s, \Delta \models B$
using *can-divide sat.simps(1) by metis*
qed
qed

lemma *combinable-star:*

assumes *combinable $\Delta \ A$*
and *combinable $\Delta \ B$*
shows *combinable $\Delta \ (\text{Star } A \ B)$*
proof (*rule combinableI-old*)
fix $a \ b \ p \ q \ x \ \sigma \ s$
assume $a, s, \Delta \models \text{Star } A \ B \wedge b, s, \Delta \models \text{Star } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd p q \odot x}$
then obtain $\text{aa } \text{ab } \text{ba } \text{bb}$ **where** $\text{Some } a = \text{aa} \oplus \text{ab} \ \text{Some } b = \text{ba} \oplus \text{bb} \ \text{aa}, s, \Delta \models A$
 $\text{ab}, s, \Delta \models B \ \text{ba}, s, \Delta \models A \ \text{bb}, s, \Delta \models B$
by *auto*
then obtain $\text{xa } \text{xb}$ **where** $\text{Some } \text{xa} = p \odot \text{aa} \oplus q \odot \text{ba} \ \text{Some } \text{xb} = p \odot \text{ab} \oplus q \odot \text{bb}$
by (*metis <a, s, $\Delta \models \text{Star } A \ B \wedge b, s, \Delta \models \text{Star } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd p q \odot x}> asso2 commutative compatible-iff compatible-multiples one-neutral option.discI option.exhaust-sel pre-logic.compatible-def$*)
then have $\text{xa}, s, \Delta \models \text{Mult } (\text{sadd p q}) \ A$
by (*meson <aa, s, $\Delta \models A$ > <ba, s, $\Delta \models A$ > assms(1) entails-def logic.combinable-def logic.sat.simps(1) logic.sat.simps(2) logic-axioms*)
moreover have $\text{xb}, s, \Delta \models \text{Mult } (\text{sadd p q}) \ B$
by (*meson <Some xb = p \odot ab \oplus q \odot bb> <ab, s, $\Delta \models B$ > <bb, s, $\Delta \models B$ > assms(2) combinable-def entails-def sat.simps(1) sat.simps(2)*)
moreover have $\text{Some } \sigma = \text{xa} \oplus \text{xb}$
using $\langle \text{Some } a = \text{aa} \oplus \text{ab} \rangle \langle \text{Some } b = \text{ba} \oplus \text{bb} \rangle \langle \text{Some } \text{xa} = p \odot \text{aa} \oplus q \odot \text{ba} \rangle \langle \text{Some } \text{xb} = p \odot \text{ab} \oplus q \odot \text{bb} \rangle \langle a, s, \Delta \models \text{Star } A \ B \wedge b, s, \Delta \models \text{Star } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd p q \odot x} \rangle$ *move-sum plus-mult* **by** *blast*
then obtain $\text{xa}' \ \text{xb}'$ **where** $\text{Some } x = \text{xa}' \oplus \text{xb}' \ \text{xa} = \text{sadd p q \odot xa}' \ \text{xb} = \text{sadd p q \odot xb}'$
by (*metis <a, s, $\Delta \models \text{Star } A \ B \wedge b, s, \Delta \models \text{Star } A \ B \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = \text{sadd p q \odot x}> plus-mult unique-inv$*)

ultimately show $x, s, \Delta \models \text{Star } A \ B$
by (*metis logic.can-divide logic-axioms sat.simps(1) sat.simps(2)*)
qed

lemma combinable-mult:
assumes *combinable* $\Delta \ A$
shows *combinable* $\Delta \ (\text{Mult } \pi \ A)$
proof (*rule combinableI*)
fix $a \ b \ p \ q \ x \ \sigma \ s$
assume *asm*: $a, s, \Delta \models \text{Mult } \pi \ A \wedge b, s, \Delta \models \text{Mult } \pi \ A \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one}$
then obtain $a' \ b'$ **where** $a', s, \Delta \models A \ b', s, \Delta \models A \ a = \pi \odot a' \ b = \pi \odot b'$ **by** *auto*

let $?p = \text{smult } p \ \pi$
let $?q = \text{smult } q \ \pi$

have $\text{Some } x = ?p \odot a' \oplus ?q \odot b'$
by (*simp add: $\langle a = \pi \odot a' \rangle \langle b = \pi \odot b' \rangle$ asm double-mult*)
moreover have $\text{sadd } ?p \ ?q = \pi$
using *asm smult-comm smult-distrib some-neutral* **by force**
ultimately show $x, s, \Delta \models \text{Mult } \pi \ A$
by (*metis $\langle a', s, \Delta \models A \rangle \langle b', s, \Delta \models A \rangle$ assms combinable-instantiate*)
qed

lemma combinable-and:
assumes *combinable* $\Delta \ A$
and *combinable* $\Delta \ B$
shows *combinable* $\Delta \ (\text{And } A \ B)$
proof (*rule combinableI*)
fix $a \ b \ p \ q \ x \ \sigma \ s$
assume $a, s, \Delta \models \text{And } A \ B \wedge b, s, \Delta \models \text{And } A \ B \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one}$
then obtain $a, s, \Delta \models A \ b, s, \Delta \models A \ a, s, \Delta \models B \ b, s, \Delta \models B$ **by** *auto*
then show $x, s, \Delta \models \text{And } A \ B$
by (*meson $\langle a, s, \Delta \models \text{And } A \ B \wedge b, s, \Delta \models \text{And } A \ B \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one} \rangle$ assms(1) assms(2) combinable-instantiate-one sat.simps(7)*)
qed

lemma combinable-forall:
assumes *combinable* $\Delta \ A$
shows *combinable* $\Delta \ (\text{Forall } x \ A)$
proof (*rule combinableI*)
fix $a \ b \ p \ q \ y \ \sigma \ s$
assume $a, s, \Delta \models \text{Forall } x \ A \wedge b, s, \Delta \models \text{Forall } x \ A \wedge \text{Some } y = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one}$
show $y, s, \Delta \models \text{Forall } x \ A$
proof (*rule sat-forall*)

fix v **show** $y, s(x := v), \Delta \models A$
by (*meson* $\langle a, s, \Delta \models \text{Forall } x A \wedge b, s, \Delta \models \text{Forall } x A \wedge \text{Some } y = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one} \rangle$ *assms combinable-instantiate-one sat.simps(9)*)
qed
qed

definition *unambiguous where*

unambiguous $\Delta A x \longleftrightarrow (\forall \sigma 1 \ \sigma 2 \ v 1 \ v 2 \ s. \ \sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models A \wedge \sigma 2, s(x := v 2), \Delta \models A \longrightarrow v 1 = v 2)$

lemma *unambiguousI:*

assumes $\bigwedge \sigma 1 \ \sigma 2 \ v 1 \ v 2 \ s. \ \sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models A \wedge \sigma 2, s(x := v 2), \Delta \models A \implies v 1 = v 2$
shows *unambiguous* $\Delta A x$
by (*simp add: assms unambiguous-def*)

lemma *unambiguous-star:*

assumes *unambiguous* $\Delta A x$
shows *unambiguous* $\Delta (\text{Star } A B) x$
proof (*rule unambiguousI*)
fix $\sigma 1 \ \sigma 2 \ v 1 \ v 2 \ s$
assume $\sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models \text{Star } A B \wedge \sigma 2, s(x := v 2), \Delta \models \text{Star } A B$
then obtain $a 1 \ b 1 \ a 2 \ b 2$ **where** *Some* $\sigma 1 = a 1 \oplus b 1$ *Some* $\sigma 2 = a 2 \oplus b 2$ $a 1, s(x := v 1), \Delta \models A$
 $a 2, s(x := v 2), \Delta \models A \ b 1, s(x := v 1), \Delta \models B \ b 2, s(x := v 2), \Delta \models B$ **by** *auto*
then have $a 1 \ \#\# \ a 2$
by (*metis* $\langle \sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models \text{Star } A B \wedge \sigma 2, s(x := v 2), \Delta \models \text{Star } A B \rangle$ *asso2 asso3 commutative*)
then show $v 1 = v 2$
using $\langle a 1, s(x := v 1), \Delta \models A \rangle \langle a 2, s(x := v 2), \Delta \models A \rangle$ *assms unambiguous-def*
by *fastforce*
qed

lemma *combinable-exists:*

assumes *combinable* ΔA
and *unambiguous* $\Delta A x$
shows *combinable* $\Delta (\text{Exists } x A)$
proof (*rule combinableI*)
fix $a \ b \ p \ q \ y \ \sigma \ s$
assume $a, s, \Delta \models \text{Exists } x A \wedge b, s, \Delta \models \text{Exists } x A \wedge \text{Some } y = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one}$
then have $a \ \#\# \ b$
by (*metis logic.compatible-multiples logic-axioms option.discI pre-logic.compatible-def*)
moreover obtain $v 1 \ v 2$ **where** $a, s(x := v 1), \Delta \models A \ b, s(x := v 2), \Delta \models A$

using $\langle a, s, \Delta \models \text{Exists } x A \wedge b, s, \Delta \models \text{Exists } x A \wedge \text{Some } y = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one} \rangle$ **by** *auto*
ultimately have $v1 = v2$
using *assms(2) unambiguous-def* **by** *force*
then show $y, s, \Delta \models \text{Exists } x A$
by (*metis (mono-tags, opaque-lifting)* $\langle a, s(x := v1), \Delta \models A \rangle$ $\langle a, s, \Delta \models \text{Exists } x A \wedge b, s, \Delta \models \text{Exists } x A \wedge \text{Some } y = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one} \rangle$ $\langle b, s(x := v2), \Delta \models A \rangle$ *assms(1) combinable-instantiate-one logic.sat.simps(8) logic-axioms*)
qed

lemma *combinable-pure*:
assumes *pure A*
shows *combinable ΔA*
using *assms combinableI-old pure-def* **by** *blast*

lemma *combinable-imp*:
assumes *pure A*
and *combinable ΔB*
shows *combinable $\Delta (\text{Imp } A B)$*
proof (*rule combinableI*)
fix $a b p q x \sigma s$
assume $a, s, \Delta \models \text{Imp } A B \wedge b, s, \Delta \models \text{Imp } A B \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one}$
then show $x, s, \Delta \models \text{Imp } A B$
using *assms(1) assms(2) combinable-instantiate-one pure-def sat.simps(5)*
by *metis*
qed

lemma *combinable-wildcard*:
assumes *combinable ΔA*
shows *combinable $\Delta (\text{Wildcard } A)$*
proof (*rule combinableI*)
fix $a b p q x \sigma s$
assume *asm: $a, s, \Delta \models \text{Wildcard } A \wedge b, s, \Delta \models \text{Wildcard } A \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p q = \text{one}$*
then obtain $a' b' pa pb$ **where** $a', s, \Delta \models A b', s, \Delta \models A a = pa \odot a' b = pb \odot b'$ **by** *auto*
then have $\text{Some } x = (\text{smult } p pa) \odot a' \oplus (\text{smult } q pb) \odot b'$
by (*simp add: asm double-mult*)
then have $x, s, \Delta \models \text{Mult } (\text{sadd } (\text{smult } p pa) (\text{smult } q pb)) A$
using $\langle a', s, \Delta \models A \rangle \langle b', s, \Delta \models A \rangle$ *assms combinable-instantiate* **by** *blast*
then show $x, s, \Delta \models \text{Wildcard } A$
by *fastforce*
qed

end

end

5 (Co)Inductive Predicates

This subsection corresponds to Section 4 of the paper [5].

theory *FixedPoint*

imports *Distributivity Combinability*

begin

type-synonym $('d, 'c, 'a)$ *chain* = *nat* \Rightarrow $('d, 'c, 'a)$ *interp*

context *logic*

begin

5.1 Definitions

definition *smaller-interp* :: $('d, 'c, 'a)$ *interp* \Rightarrow $('d, 'c, 'a)$ *interp* \Rightarrow *bool* **where**
smaller-interp Δ Δ' \longleftrightarrow $(\forall s. \Delta s \subseteq \Delta' s)$

lemma *smaller-interpI*:

assumes $\bigwedge s x. x \in \Delta s \Longrightarrow x \in \Delta' s$

shows *smaller-interp* Δ Δ'

by (*simp add: assms smaller-interp-def subsetI*)

definition *indep-interp* **where**

indep-interp $A \longleftrightarrow (\forall x s \Delta \Delta'. x, s, \Delta \models A \longleftrightarrow x, s, \Delta' \models A)$

fun *applies-eq* :: $('a, 'b, 'c, 'd)$ *assertion* \Rightarrow $('d, 'c, 'a)$ *interp* \Rightarrow $('d, 'c, 'a)$ *interp*
where

applies-eq $A \Delta s = \{ a \mid a, s, \Delta \models A \}$

definition *monotonic* :: $(('d, 'c, 'a)$ *interp* \Rightarrow $('d, 'c, 'a)$ *interp*) \Rightarrow *bool* **where**
monotonic $f \longleftrightarrow (\forall \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \longrightarrow \text{smaller-interp } (f \Delta) (f \Delta'))$

lemma *monotonicI*:

assumes $\bigwedge \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \Longrightarrow \text{smaller-interp } (f \Delta) (f \Delta')$

shows *monotonic* f

by (*simp add: assms monotonic-def*)

definition *non-increasing* :: $(('d, 'c, 'a)$ *interp* \Rightarrow $('d, 'c, 'a)$ *interp*) \Rightarrow *bool* **where**
non-increasing $f \longleftrightarrow (\forall \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \longrightarrow \text{smaller-interp } (f \Delta') (f \Delta))$

lemma *non-increasingI*:

assumes $\bigwedge \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \Longrightarrow \text{smaller-interp } (f \Delta') (f \Delta)$

shows *non-increasing* f

by (simp add: assms non-increasing-def)

lemma *smaller-interp-refl*:
smaller-interp Δ Δ
by (simp add: smaller-interp-def)

lemma *smaller-interp-applies-cons*:
assumes smaller-interp (applies-eq A Δ) (applies-eq A Δ')
and $a, s, \Delta \models A$
shows $a, s, \Delta' \models A$
proof –
have $a \in$ applies-eq A Δ s
using assms(2) by force
then have $a \in$ applies-eq A Δ' s
by (metis assms(1) in-mono smaller-interp-def)
then show ?thesis by auto
qed

definition *empty-interp* where
empty-interp $s = \{\}$

definition *full-interp* :: ($'d, 'c, 'a$) interp where
full-interp $s = UNIV$

lemma *smaller-interp-trans*:
assumes smaller-interp a b
and smaller-interp b c
shows smaller-interp a c
by (metis assms(1) assms(2) dual-order.trans smaller-interp-def)

lemma *smaller-empty*:
smaller-interp empty-interp x
by (simp add: empty-interp-def smaller-interp-def)

The definition of set-closure properties corresponds to Definition 8 of the paper [5].

definition *set-closure-property* :: ($'a \Rightarrow 'a \Rightarrow 'a$ set) \Rightarrow ($'d, 'c, 'a$) interp \Rightarrow bool
where
set-closure-property S $\Delta \iff (\forall a b s. a \in \Delta s \wedge b \in \Delta s \longrightarrow S a b \subseteq \Delta s)$

lemma *set-closure-propertyI*:
assumes $\bigwedge a b s. a \in \Delta s \wedge b \in \Delta s \implies S a b \subseteq \Delta s$
shows set-closure-property S Δ
by (simp add: assms set-closure-property-def)

lemma *set-closure-property-instantiate*:
assumes set-closure-property S Δ

and $a \in \Delta s$
and $b \in \Delta s$
and $x \in S a b$
shows $x \in \Delta s$
using *assms subsetD set-closure-property-def* **by** *metis*

5.2 Everything preserves monotonicity

lemma *indep-implies-non-increasing*:
assumes *indep-interp A*
shows *non-increasing (applies-eq A)*
by (*metis (no-types, lifting) applies-eq.simps assms indep-interp-def smaller-interp-def mem-Collect-eq non-increasingI subsetI*)

5.2.1 Monotonicity

lemma *mono-instantiate*:
assumes *monotonic (applies-eq A)*
and $x \in \text{applies-eq } A \Delta s$
and *smaller-interp $\Delta \Delta'$*
shows $x \in \text{applies-eq } A \Delta' s$
using *assms(1) assms(2) assms(3) monotonic-def smaller-interp-applies-cons*
by *fastforce*

lemma *mono-star*:
assumes *monotonic (applies-eq A)*
and *monotonic (applies-eq B)*
shows *monotonic (applies-eq (Star A B))*
proof (*rule monotonicI*)
fix $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$
assume *asm0: smaller-interp $\Delta \Delta'$*
show *smaller-interp (applies-eq (Star A B) Δ) (applies-eq (Star A B) Δ')*
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1: $x \in \text{applies-eq (Star A B) } \Delta s$*
then obtain $a b$ **where** *Some $x = a \oplus b$ $a \in \text{applies-eq } A \Delta s$ $b \in \text{applies-eq } B \Delta s$*
by *auto*
then have $a \in \text{applies-eq } A \Delta' s \wedge b \in \text{applies-eq } B \Delta' s$
by (*meson asm0 assms(1) assms(2) mono-instantiate*)
then show $x \in \text{applies-eq (Star A B) } \Delta' s$
using $\langle \text{Some } x = a \oplus b \rangle$ **by** *force*
qed
qed

lemma *mono-wand*:
assumes *non-increasing (applies-eq A)*
and *monotonic (applies-eq B)*
shows *monotonic (applies-eq (Wand A B))*
proof (*rule monotonicI*)

```

fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
assume  $\text{asm0}: \text{smaller-interp } \Delta \Delta'$ 
show  $\text{smaller-interp } (\text{applies-eq } (\text{Wand } A \ B) \ \Delta) (\text{applies-eq } (\text{Wand } A \ B) \ \Delta')$ 
proof (rule smaller-interpI)
  fix  $s \ x$  assume  $\text{asm1}: x \in \text{applies-eq } (\text{Wand } A \ B) \ \Delta \ s$ 
  have  $x, s, \Delta' \models \text{Wand } A \ B$ 
  proof (rule sat-wand)
    fix  $a \ b$ 
    assume  $\text{asm2}: a, s, \Delta' \models A \wedge \text{Some } b = x \oplus a$ 
    then have  $a, s, \Delta \models A$ 
      by (meson asm0 assms(1) non-increasing-def smaller-interp-applies-cons)
    then have  $b, s, \Delta \models B$ 
      using  $\text{asm1 } \text{asm2}$  by auto
    then show  $b, s, \Delta' \models B$ 
      by (meson asm0 assms(2) monotonic-def smaller-interp-applies-cons)
  qed
then show  $x \in \text{applies-eq } (\text{Wand } A \ B) \ \Delta' \ s$ 
  by simp
qed
qed

```

lemma *mono-and*:

```

assumes monotonic (applies-eq  $A$ )
and monotonic (applies-eq  $B$ )
shows monotonic (applies-eq ( $\text{And } A \ B$ ))
proof (rule monotonicI)
fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
assume  $\text{asm0}: \text{smaller-interp } \Delta \Delta'$ 
show  $\text{smaller-interp } (\text{applies-eq } (\text{And } A \ B) \ \Delta) (\text{applies-eq } (\text{And } A \ B) \ \Delta')$ 
proof (rule smaller-interpI)
  fix  $s \ x$  assume  $\text{asm1}: x \in \text{applies-eq } (\text{And } A \ B) \ \Delta \ s$ 
  then show  $x \in \text{applies-eq } (\text{And } A \ B) \ \Delta' \ s$ 
    using  $\text{asm0 } \text{assms}(1) \ \text{assms}(2)$  monotonic-def logic-axioms mem-Collect-eq
    sat.simps(8) smaller-interp-applies-cons by fastforce
  qed
qed

```

lemma *mono-or*:

```

assumes monotonic (applies-eq  $A$ )
and monotonic (applies-eq  $B$ )
shows monotonic (applies-eq ( $\text{Or } A \ B$ ))
proof (rule monotonicI)
fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
assume  $\text{asm0}: \text{smaller-interp } \Delta \Delta'$ 
show  $\text{smaller-interp } (\text{applies-eq } (\text{Or } A \ B) \ \Delta) (\text{applies-eq } (\text{Or } A \ B) \ \Delta')$ 
proof (rule smaller-interpI)
  fix  $s \ x$  assume  $\text{asm1}: x \in \text{applies-eq } (\text{Or } A \ B) \ \Delta \ s$ 

```

```

    then show  $x \in \text{applies-eq } (Or A B) \Delta' s$ 
      using asm0 assms(1) assms(2) monotonic-def logic-axioms mem-Collect-eq
sat.simps(8) smaller-interp-applies-cons by fastforce
    qed
  qed

```

```

lemma mono-sem:
  monotonic (applies-eq (Sem B))
  using monotonic-def smaller-interp-def by fastforce

```

```

lemma mono-interp:
  monotonic (applies-eq Pred)
proof (rule monotonicI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq Pred  $\Delta)$  (applies-eq Pred  $\Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume  $x \in \text{applies-eq Pred } \Delta s$ 
    then show  $x \in \text{applies-eq Pred } \Delta' s$ 
      by (metis (mono-tags, lifting) <smaller-interp  $\Delta \Delta'$ ) applies-eq.simps in-mono
mem-Collect-eq sat.simps(10) smaller-interp-def)
  qed
qed

```

```

lemma mono-mult:
  assumes monotonic (applies-eq A)
  shows monotonic (applies-eq (Mult  $\pi A)$ )
proof (rule monotonicI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Mult  $\pi A) \Delta)$  (applies-eq (Mult  $\pi A) \Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq (Mult } \pi A) \Delta s$ 
    then show  $x \in \text{applies-eq (Mult } \pi A) \Delta' s$ 
      using asm0 assms monotonic-def smaller-interp-applies-cons by fastforce
  qed
qed

```

```

lemma mono-wild:
  assumes monotonic (applies-eq A)
  shows monotonic (applies-eq (Wildcard A))
proof (rule monotonicI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Wildcard A) \Delta) (applies-eq (Wildcard A) \Delta')
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq (Wildcard A) } \Delta s$ 
    then show  $x \in \text{applies-eq (Wildcard A) } \Delta' s$ 

```

```

    using asm0 assms monotonic-def smaller-interp-applies-cons by fastforce
  qed
qed

```

lemma *mono-imp*:

```

  assumes non-increasing (applies-eq A)
    and monotonic (applies-eq B)
  shows monotonic (applies-eq (Imp A B))
proof (rule monotonicI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Imp A B)  $\Delta$ ) (applies-eq (Imp A B)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq (Imp A B) } \Delta s$ 
    have  $x, s, \Delta' \models \text{Imp A B}$ 
    proof (cases  $x, s, \Delta' \models A$ )
      case True
      then have  $x, s, \Delta \models A$ 
        by (meson asm0 assms(1) non-increasing-def smaller-interp-applies-cons)
      then have  $x, s, \Delta \models B$ 
        using asm1 by auto
      then show ?thesis
        by (metis asm0 assms(2) monotonic-def sat.simps(5) smaller-interp-applies-cons)
    next
      case False
      then show ?thesis by simp
    qed
  then show  $x \in \text{applies-eq (Imp A B) } \Delta' s$ 
    by simp
  qed
qed

```

lemma *mono-bounded*:

```

  assumes monotonic (applies-eq A)
  shows monotonic (applies-eq (Bounded A))
proof (rule monotonicI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Bounded A)  $\Delta$ ) (applies-eq (Bounded A)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume  $x \in \text{applies-eq (Bounded A) } \Delta s$ 
    then show  $x \in \text{applies-eq (Bounded A) } \Delta' s$ 
      using asm assms monotonic-def smaller-interp-applies-cons by fastforce
    qed
  qed

```

lemma *mono-exists*:

```

  assumes monotonic (applies-eq A)

```

shows *monotonic* (*applies-eq* (*Exists v A*))
proof (*rule monotonicI*)
fix $\Delta \Delta' :: ('c, 'd, 'a)$ *interp*
assume *asm0*: *smaller-interp* $\Delta \Delta'$
show *smaller-interp* (*applies-eq* (*Exists v A*) Δ) (*applies-eq* (*Exists v A*) Δ')
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1*: $x \in$ *applies-eq* (*Exists v A*) Δs
then show $x \in$ *applies-eq* (*Exists v A*) $\Delta' s$
using *asm0 assms monotonic-def smaller-interp-applies-cons* **by** *fastforce*
qed
qed

lemma *mono-forall*:
assumes *monotonic* (*applies-eq A*)
shows *monotonic* (*applies-eq* (*Forall v A*))
proof (*rule monotonicI*)
fix $\Delta \Delta' :: ('c, 'd, 'a)$ *interp*
assume *asm0*: *smaller-interp* $\Delta \Delta'$
show *smaller-interp* (*applies-eq* (*Forall v A*) Δ) (*applies-eq* (*Forall v A*) Δ')
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1*: $x \in$ *applies-eq* (*Forall v A*) Δs
then show $x \in$ *applies-eq* (*Forall v A*) $\Delta' s$
using *asm0 assms monotonic-def smaller-interp-applies-cons* **by** *fastforce*
qed
qed

5.2.2 Non-increasing

lemma *non-increasing-instantiate*:
assumes *non-increasing* (*applies-eq A*)
and $x \in$ *applies-eq A* $\Delta' s$
and *smaller-interp* $\Delta \Delta'$
shows $x \in$ *applies-eq A* Δs
using *assms(1) assms(2) assms(3) non-increasing-def smaller-interp-applies-cons*
by *fastforce*

lemma *non-inc-star*:
assumes *non-increasing* (*applies-eq A*)
and *non-increasing* (*applies-eq B*)
shows *non-increasing* (*applies-eq* (*Star A B*))
proof (*rule non-increasingI*)
fix $\Delta \Delta' :: ('c, 'd, 'a)$ *interp*
assume *asm0*: *smaller-interp* $\Delta \Delta'$
show *smaller-interp* (*applies-eq* (*Star A B*) Δ') (*applies-eq* (*Star A B*) Δ)
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1*: $x \in$ *applies-eq* (*Star A B*) $\Delta' s$
then obtain $a b$ **where** *Some* $x = a \oplus b$ $a \in$ *applies-eq A* $\Delta' s$ $b \in$ *applies-eq B* $\Delta' s$

```

    by auto
  then have  $a \in \text{applies-eq } A \Delta s \wedge b \in \text{applies-eq } B \Delta s$ 
    by (meson asm0 assms(1) assms(2) non-increasing-instantiate)
  then show  $x \in \text{applies-eq } (\text{Star } A B) \Delta s$ 
    using ‹Some  $x = a \oplus b$ › by force
qed
qed

```

```

lemma non-increasing-wand:
  assumes monotonic (applies-eq A)
    and non-increasing (applies-eq B)
  shows non-increasing (applies-eq (Wand A B))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Wand A B)  $\Delta'$ ) (applies-eq (Wand A B)  $\Delta$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq } (\text{Wand } A B) \Delta' s$ 
    have  $x, s, \Delta \models \text{Wand } A B$ 
    proof (rule sat-wand)
      fix  $a b$ 
      assume asm2:  $a, s, \Delta \models A \wedge \text{Some } b = x \oplus a$ 
      then have  $a, s, \Delta' \models A$ 
        by (meson asm0 assms(1) monotonic-def smaller-interp-applies-cons)
      then have  $b, s, \Delta' \models B$ 
        using asm1 asm2 by auto
      then show  $b, s, \Delta \models B$ 
        by (meson asm0 assms(2) non-increasing-def smaller-interp-applies-cons)
    qed
  then show  $x \in \text{applies-eq } (\text{Wand } A B) \Delta s$ 
    by simp
  qed
qed

```

```

lemma non-increasing-and:
  assumes non-increasing (applies-eq A)
    and non-increasing (applies-eq B)
  shows non-increasing (applies-eq (And A B))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta' \Delta$ 
  show smaller-interp (applies-eq (And A B)  $\Delta$ ) (applies-eq (And A B)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq } (\text{And } A B) \Delta s$ 
    then show  $x \in \text{applies-eq } (\text{And } A B) \Delta' s$ 
      using asm0 assms(1) assms(2) non-increasing-def logic-axioms mem-Collect-eq

```

sat.simps(8) smaller-interp-applies-cons **by** *fastforce*
qed
qed

lemma *non-increasing-or*:
assumes *non-increasing (applies-eq A)*
and *non-increasing (applies-eq B)*
shows *non-increasing (applies-eq (Or A B))*
proof (*rule non-increasingI*)
fix $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$
assume *asm0: smaller-interp $\Delta \Delta'$*
show *smaller-interp (applies-eq (Or A B) Δ') (applies-eq (Or A B) Δ)*
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1: $x \in \text{applies-eq (Or A B) } \Delta' s$*
then show *$x \in \text{applies-eq (Or A B) } \Delta s$*
using *asm0 assms(1) assms(2) non-increasing-def logic-axioms mem-Collect-eq*
sat.simps(8) smaller-interp-applies-cons **by** *fastforce*
qed
qed

lemma *non-increasing-sem*:
non-increasing (applies-eq (Sem B))
using *non-increasing-def smaller-interp-def* **by** *fastforce*

lemma *non-increasing-mult*:
assumes *non-increasing (applies-eq A)*
shows *non-increasing (applies-eq (Mult π A))*
proof (*rule non-increasingI*)
fix $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$
assume *asm0: smaller-interp $\Delta \Delta'$*
show *smaller-interp (applies-eq (Mult π A) Δ') (applies-eq (Mult π A) Δ)*
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1: $x \in \text{applies-eq (Mult } \pi \text{ A) } \Delta' s$*
then show *$x \in \text{applies-eq (Mult } \pi \text{ A) } \Delta s$*
using *asm0 assms non-increasing-def smaller-interp-applies-cons* **by** *fastforce*
qed
qed

lemma *non-increasing-wild*:
assumes *non-increasing (applies-eq A)*
shows *non-increasing (applies-eq (Wildcard A))*
proof (*rule non-increasingI*)
fix $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$
assume *asm0: smaller-interp $\Delta \Delta'$*
show *smaller-interp (applies-eq (Wildcard A) Δ') (applies-eq (Wildcard A) Δ)*
proof (*rule smaller-interpI*)
fix $s x$ **assume** *asm1: $x \in \text{applies-eq (Wildcard A) } \Delta' s$*
then show *$x \in \text{applies-eq (Wildcard A) } \Delta s$*

```

    using asm0 assms non-increasing-def smaller-interp-applies-cons by fastforce
  qed
qed

```

```

lemma non-increasing-imp:
  assumes monotonic (applies-eq A)
    and non-increasing (applies-eq B)
  shows non-increasing (applies-eq (Imp A B))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm0: smaller-interp  $\Delta \Delta'$ 
  show smaller-interp (applies-eq (Imp A B)  $\Delta'$ ) (applies-eq (Imp A B)  $\Delta$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume asm1:  $x \in \text{applies-eq (Imp A B) } \Delta' s$ 
    have  $x, s, \Delta \models \text{Imp A B}$ 
    proof (cases  $x, s, \Delta \models A$ )
      case True
      then have  $x, s, \Delta' \models A$ 
        by (meson asm0 assms(1) monotonic-def smaller-interp-applies-cons)
      then have  $x, s, \Delta' \models B$ 
        using asm1 by auto
      then show ?thesis
        by (metis asm0 assms(2) non-increasing-def sat.simps(5) smaller-interp-applies-cons)
    next
      case False
      then show ?thesis by simp
    qed
  then show  $x \in \text{applies-eq (Imp A B) } \Delta s$ 
    by simp
  qed
qed

```

```

lemma non-increasing-bounded:
  assumes non-increasing (applies-eq A)
  shows non-increasing (applies-eq (Bounded A))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a) \text{interp}$ 
  assume asm: smaller-interp  $\Delta' \Delta$ 
  show smaller-interp (applies-eq (Bounded A)  $\Delta$ ) (applies-eq (Bounded A)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix  $s x$  assume  $x \in \text{applies-eq (Bounded A) } \Delta s$ 
    then show  $x \in \text{applies-eq (Bounded A) } \Delta' s$ 
      using asm assms non-increasing-def smaller-interp-applies-cons by fastforce
    qed
  qed

```

```

lemma non-increasing-exists:
  assumes non-increasing (applies-eq A)
  shows non-increasing (applies-eq (Exists v A))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a)$  interp
  assume asm0: smaller-interp  $\Delta' \Delta$ 
  show smaller-interp (applies-eq (Exists v A)  $\Delta$ ) (applies-eq (Exists v A)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix s x assume asm1:  $x \in$  applies-eq (Exists v A)  $\Delta$  s
    then show  $x \in$  applies-eq (Exists v A)  $\Delta'$  s
      using asm0 assms non-increasing-def smaller-interp-applies-cons by fastforce
    qed
  qed

```

```

lemma non-increasing-forall:
  assumes non-increasing (applies-eq A)
  shows non-increasing (applies-eq (Forall v A))
proof (rule non-increasingI)
  fix  $\Delta \Delta' :: ('c, 'd, 'a)$  interp
  assume asm0: smaller-interp  $\Delta' \Delta$ 
  show smaller-interp (applies-eq (Forall v A)  $\Delta$ ) (applies-eq (Forall v A)  $\Delta'$ )
  proof (rule smaller-interpI)
    fix s x assume asm1:  $x \in$  applies-eq (Forall v A)  $\Delta$  s
    then show  $x \in$  applies-eq (Forall v A)  $\Delta'$  s
      using asm0 assms non-increasing-def smaller-interp-applies-cons by fastforce
    qed
  qed

```

5.3 Tarski's fixed points

5.3.1 Greatest Fixed Point

definition $D :: (('d, 'c, 'a)$ *interp* \Rightarrow (*'d, 'c, 'a*) *interp*) \Rightarrow (*'d, 'c, 'a*) *interp* **set**
where

$$D f = \{ \Delta \mid \Delta. \text{smaller-interp } \Delta (f \Delta) \}$$

fun *GFP* :: (('d, 'c, 'a) *interp* \Rightarrow (*'d, 'c, 'a*) *interp*) \Rightarrow (*'d, 'c, 'a*) *interp* **where**
GFP f s = { $\sigma \mid \sigma. \exists \Delta \in D f. \sigma \in \Delta s$ }

lemma *smaller-interp-D*:

assumes $x \in D f$

shows *smaller-interp* x (*GFP* f)

by (*metis (mono-tags, lifting) CollectI GFP.elims assms smaller-interpI*)

lemma *GFP-lub*:

assumes $\bigwedge x. x \in D f \implies$ *smaller-interp* x y

shows *smaller-interp* (*GFP* f) y

proof (*rule smaller-interpI*)

fix s x

```

assume  $x \in GFP\ f\ s$ 
then obtain  $\Delta$  where  $\Delta \in D\ f\ x \in \Delta\ s$ 
  by auto
then show  $x \in y\ s$ 
  by (metis assms in-mono smaller-interp-def)
qed

```

```

lemma smaller-interp-antisym:
  assumes smaller-interp a b
    and smaller-interp b a
  shows  $a = b$ 
proof (rule ext)
  fix  $x$  show  $a\ x = b\ x$ 
  by (metis assms(1) assms(2) set-eq-subset smaller-interp-def)
qed

```

5.3.2 Least Fixed Point

definition $DD :: (('d, 'c, 'a)\ interp \Rightarrow ('d, 'c, 'a)\ interp) \Rightarrow ('d, 'c, 'a)\ interp\ set$
where

$$DD\ f = \{ \Delta \mid \Delta.\ smaller_interp\ (f\ \Delta)\ \Delta \}$$

fun $LFP :: (('d, 'c, 'a)\ interp \Rightarrow ('d, 'c, 'a)\ interp) \Rightarrow ('d, 'c, 'a)\ interp$ **where**
 $LFP\ f\ s = \{ \sigma \mid \sigma.\ \forall \Delta \in DD\ f.\ \sigma \in \Delta\ s \}$

```

lemma smaller-interp-DD:
  assumes  $x \in DD\ f$ 
  shows smaller-interp (LFP f) x
  using assms smaller-interp-def by fastforce

```

```

lemma LFP-glb:
  assumes  $\bigwedge x.\ x \in DD\ f \Longrightarrow smaller\_interp\ y\ x$ 
  shows smaller-interp y (LFP f)
proof (rule smaller-interpI)
  fix  $s\ x$ 
  assume  $x \in y\ s$ 
  then have  $\bigwedge \Delta.\ \Delta \in DD\ f \Longrightarrow x \in \Delta\ s$ 
    by (metis assms smaller-interp-def subsetD)
  then show  $x \in LFP\ f\ s$ 
    by simp
qed

```

5.4 Combinability and (an assertion being) intuitionistic are set-closure properties

5.4.1 Intuitionistic assertions

definition $sem_intui :: (('d, 'c, 'a)\ interp \Rightarrow bool)$ **where**
 $sem_intui\ \Delta \longleftrightarrow (\forall s\ \sigma\ \sigma'.\ \sigma' \succeq \sigma \wedge \sigma \in \Delta\ s \longrightarrow \sigma' \in \Delta\ s)$

```

lemma sem-intuiI:
  assumes  $\bigwedge s \sigma \sigma'. \sigma' \succeq \sigma \wedge \sigma \in \Delta s \implies \sigma' \in \Delta s$ 
  shows sem-intui  $\Delta$ 
  using assms sem-intui-def by blast

lemma instantiate-intui-applies:
  assumes intuitionistic  $s \Delta A$ 
    and  $\sigma' \succeq \sigma$ 
    and  $\sigma \in \text{applies-eq } A \Delta s$ 
  shows  $\sigma' \in \text{applies-eq } A \Delta s$ 
  using assms(1) assms(2) assms(3) intuitionistic-def by fastforce

lemma sem-intui-intuitionistic:
  sem-intui (applies-eq  $A \Delta$ )  $\longleftrightarrow$  ( $\forall s. \text{intuitionistic } s \Delta A$ ) (is  $?A \longleftrightarrow ?B$ )
proof
  show  $?B \implies ?A$ 
  proof -
    assume  $?B$ 
    show  $?A$ 
    proof (rule sem-intuiI)
      fix  $s \sigma \sigma'$ 
      assume  $\sigma' \succeq \sigma \wedge \sigma \in \text{applies-eq } A \Delta s$ 
      then show  $\sigma' \in \text{applies-eq } A \Delta s$ 
      using  $\langle \forall s. \text{intuitionistic } s \Delta A \rangle$  instantiate-intui-applies by blast
    qed
  qed
  assume  $?A$ 
  show  $?B$ 
  proof
    fix  $s$  show intuitionistic  $s \Delta A$ 
    proof (rule intuitionisticI)
      fix  $a b$ 
      assume  $a \succeq b \wedge b, s, \Delta \models A$ 
      then have  $b \in \text{applies-eq } A \Delta s$  by simp
      then show  $a, s, \Delta \models A$ 
      by (metis CollectD  $\langle a \succeq b \wedge b, s, \Delta \models A \rangle$   $\langle \text{sem-intui } (\text{applies-eq } A \Delta) \rangle$ 
applies-eq.simps sem-intui-def)
    qed
  qed
qed

```

```

lemma intuitionistic-set-closure:
  sem-intui = set-closure-property ( $\lambda a b. \{ \sigma \mid \sigma. \sigma \succeq a \}$ )
proof (rule ext)
  fix  $\Delta :: ('c, 'd, 'a) \text{interp}$ 
  show sem-intui  $\Delta$  = set-closure-property ( $\lambda a b. \{ \sigma \mid \sigma. \sigma \succeq a \}$ )  $\Delta$  (is  $?A \longleftrightarrow$ 

```

?B)
proof
show ?A \implies ?B
by (*metis* (*no-types*, *lifting*) *CollectD set-closure-propertyI sem-intui-def sub-setI*)
assume ?B
show ?A
proof (*rule sem-intuiI*)
fix s σ σ'
assume $\sigma' \succeq \sigma \wedge \sigma \in \Delta s$
moreover have $(\lambda a b. \{\sigma \mid \sigma. \sigma \succeq a\}) \sigma \sigma = \{\sigma' \mid \sigma'. \sigma' \succeq \sigma\}$ **by** *simp*
ultimately have $\{\sigma' \mid \sigma'. \sigma' \succeq \sigma\} \subseteq \Delta s$
by (*metis* \langle *set-closure-property* $(\lambda a b. \{\sigma \mid \sigma. \sigma \succeq a\}) \Delta \rangle$ *set-closure-property-def*)
show $\sigma' \in \Delta s$
using $\langle \sigma' \succeq \sigma \wedge \sigma \in \Delta s \rangle \langle \{\sigma' \mid \sigma'. \sigma' \succeq \sigma\} \subseteq \Delta s \rangle$ **by** *fastforce*
qed
qed
qed

5.4.2 Combinable assertions

definition *sem-combinable* :: ('d, 'c, 'a) *interp* \Rightarrow *bool* **where**
sem-combinable $\Delta \longleftrightarrow (\forall s p q a b x. \text{sadd } p q = \text{one} \wedge a \in \Delta s \wedge b \in \Delta s \wedge$
Some $x = p \odot a \oplus q \odot b \longrightarrow x \in \Delta s)$

lemma *sem-combinableI*:
assumes $\bigwedge s p q a b x. \text{sadd } p q = \text{one} \wedge a \in \Delta s \wedge b \in \Delta s \wedge \text{Some } x = p \odot$
 $a \oplus q \odot b \implies x \in \Delta s$
shows *sem-combinable* Δ
using *assms sem-combinable-def* **by** *blast*

lemma *sem-combinableE*:
assumes *sem-combinable* Δ
and $a \in \Delta s$
and $b \in \Delta s$
and $\text{Some } x = p \odot a \oplus q \odot b$
and $\text{sadd } p q = \text{one}$
shows $x \in \Delta s$
using *assms(1) assms(2) assms(3) assms(4) assms(5) sem-combinable-def*[of
 $\Delta]$
by *blast*

lemma *applies-eq-equiv*:
 $x \in \text{applies-eq } A \Delta s \longleftrightarrow x, s, \Delta \models A$
by *simp*

lemma *sem-combinable-appliesE*:
assumes *sem-combinable* (*applies-eq* $A \Delta$)
and $a, s, \Delta \models A$

and $b, s, \Delta \models A$
and $\text{Some } x = p \odot a \oplus q \odot b$
and $\text{sadd } p \ q = \text{one}$
shows $x, s, \Delta \models A$
using sem-combinableE [of $\text{applies-eq } A \ \Delta \ a \ s \ b \ x \ p \ q$] *assms* **by** *simp*

lemma *sem-combinable-equiv*:

$\text{sem-combinable } (\text{applies-eq } A \ \Delta) \longleftrightarrow (\text{combinable } \Delta \ A) \ (\text{is } ?A \longleftrightarrow ?B)$

proof

show $?B \implies ?A$

proof –

assume $?B$

show $?A$

proof (*rule sem-combinableI*)

fix $s \ p \ q \ a \ b \ x$

assume $\text{asm}: \text{sadd } p \ q = \text{one} \wedge a \in \text{applies-eq } A \ \Delta \ s \wedge b \in \text{applies-eq } A \ \Delta \ s$
 $\wedge \text{Some } x = p \odot a \oplus q \odot b$

then show $x \in \text{applies-eq } A \ \Delta \ s$

using $\langle \text{combinable } \Delta \ A \rangle \text{applies-eq-equiv combinable-instantiate-one}$ **by** *blast*

qed

qed

assume $?A$

show $?B$

proof –

fix s **show** $\text{combinable } \Delta \ A$

proof (*rule combinableI*)

fix $a \ b \ p \ q \ x \ \sigma \ s$

assume $a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one}$

then show $x, s, \Delta \models A$

using $\langle \text{sem-combinable } (\text{applies-eq } A \ \Delta) \rangle \text{sem-combinable-appliesE}$ **by** *blast*

qed

qed

qed

lemma *combinable-set-closure*:

$\text{sem-combinable} = \text{set-closure-property } (\lambda a \ b. \{ \sigma \mid \sigma \ p \ q. \text{sadd } p \ q = \text{one} \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \})$

proof (*rule ext*)

fix $\Delta :: ('c, 'd, 'a) \text{interp}$

show $\text{sem-combinable } \Delta = \text{set-closure-property } (\lambda a \ b. \{ \sigma \mid \sigma \ p \ q. \text{sadd } p \ q = \text{one} \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \}) \ \Delta \ (\text{is } ?A \longleftrightarrow ?B)$

proof

show $?A \implies ?B$

proof –

assume $?A$

show $?B$

proof (*rule set-closure-propertyI*)

```

fix a b s
assume a ∈ Δ s ∧ b ∈ Δ s
then show {x. ∃σ p q. x = σ ∧ sadd p q = one ∧ Some σ = p ⊙ a ⊕ q ⊙
b} ⊆ Δ s
  using ⟨sem-combinable Δ⟩ sem-combinableE by blast
qed
qed
assume ?B
show ?A
proof (rule sem-combinableI)
  fix s p q a b x
  assume asm: sadd p q = one ∧ a ∈ Δ s ∧ b ∈ Δ s ∧ Some x = p ⊙ a ⊕ q
⊙ b

  then have x ∈ (λa b. { σ |σ p q. sadd p q = one ∧ Some σ = p ⊙ a ⊕ q ⊙
b}) a b
  by blast
  moreover have (λa b. { σ |σ p q. sadd p q = one ∧ Some σ = p ⊙ a ⊕ q
⊙ b}) a b ⊆ Δ s
  using ⟨?B⟩ set-closure-property-def[of (λa b. { σ |σ p q. sadd p q = one ∧
Some σ = p ⊙ a ⊕ q ⊙ b}) Δ]
  asm by meson
  ultimately show x ∈ Δ s by blast
qed
qed
qed

```

5.5 Transfinite induction

definition *Inf* :: ('d, 'c, 'a) interp set ⇒ ('d, 'c, 'a) interp **where**
Inf S s = { σ |σ. ∀Δ ∈ S. σ ∈ Δ s }

definition *Sup* :: ('d, 'c, 'a) interp set ⇒ ('d, 'c, 'a) interp **where**
Sup S s = { σ |σ. ∃Δ ∈ S. σ ∈ Δ s }

definition *inf* :: ('d, 'c, 'a) interp ⇒ ('d, 'c, 'a) interp ⇒ ('d, 'c, 'a) interp **where**
inf Δ Δ' s = Δ s ∩ Δ' s

definition *less* **where**
less a b ⇔ smaller-interp a b ∧ a ≠ b

definition *sup* :: ('d, 'c, 'a) interp ⇒ ('d, 'c, 'a) interp ⇒ ('d, 'c, 'a) interp **where**
sup Δ Δ' s = Δ s ∪ Δ' s

lemma *smaller-full*:
smaller-interp x full-interp
by (simp add: full-interp-def smaller-interpI)

lemma *inf-empty*:

local.Inf {} = *full-interp*

proof (*rule ext*)

fix *s* :: 'c \Rightarrow 'd **show** *local.Inf* {} *s* = *full-interp* *s*

by (*simp add: Inf-def full-interp-def*)

qed

lemma *sup-empty*:

local.Sup {} = *empty-interp*

proof (*rule ext*)

fix *s* :: 'c \Rightarrow 'd **show** *local.Sup* {} *s* = *empty-interp* *s*

by (*simp add: Sup-def empty-interp-def*)

qed

lemma *test-axiom-inf*:

assumes $\bigwedge x. x \in A \Longrightarrow$ *smaller-interp* *z* *x*

shows *smaller-interp* *z* (*local.Inf* *A*)

proof (*rule smaller-interpI*)

fix *s* *x*

assume $x \in z$ *s*

then have $\bigwedge y. y \in A \Longrightarrow x \in y$ *s*

by (*metis assms in-mono smaller-interp-def*)

then show $x \in$ *local.Inf* *A* *s*

by (*simp add: Inf-def*)

qed

lemma *test-axiom-sup*:

assumes $\bigwedge x. x \in A \Longrightarrow$ *smaller-interp* *x* *z*

shows *smaller-interp* (*local.Sup* *A*) *z*

proof (*rule smaller-interpI*)

fix *s* *x*

assume $x \in$ *local.Sup* *A* *s*

then obtain *y* **where** $y \in A$ $x \in y$ *s*

using *Sup-def*[*of A s*] *mem-Collect-eq*[*of x*]

by *auto*

then show $x \in z$ *s*

by (*metis assms smaller-interp-def subsetD*)

qed

interpretation *complete-lattice* *Inf* *Sup* *inf* *smaller-interp* *less* *sup* *empty-interp* *full-interp*

apply *standard*

apply (*metis less-def smaller-interp-antisym*)

apply (*simp add: smaller-interp-refl*)

using *smaller-interp-trans* **apply** *blast*

using *smaller-interp-antisym* **apply** *blast*

apply (*simp add: inf-def smaller-interp-def*)

```

apply (simp add: inf-def smaller-interp-def)
apply (simp add: inf-def smaller-interp-def)
apply (simp add: smaller-interpI sup-def)
apply (simp add: smaller-interpI sup-def)
apply (simp add: smaller-interp-def sup-def)
apply (metis (mono-tags, lifting) CollectD Inf-def smaller-interpI)
using test-axiom-inf apply blast
apply (metis (mono-tags, lifting) CollectI Sup-def smaller-interpI)
using test-axiom-sup apply auto[1]
apply (simp add: inf-empty)
by (simp add: sup-empty)

```

lemma *mono-same*:

```

  monotonic f  $\longleftrightarrow$  order-class.mono f
by (metis (no-types, opaque-lifting) le-funE le-funI monotonic-def order-class.mono-def
  smaller-interp-def)

```

lemma *smaller-interp a b \longleftrightarrow a \leq b*

```

by (simp add: le-fun-def smaller-interp-def)

```

lemma *set-closure-property-admissible*:

```

  ccpo.admissible Sup-class.Sup ( $\leq$ ) (set-closure-property S)
proof (rule ccpo.admissibleI)
fix A :: ('c, 'd, 'a) interp set
assume asm0: Complete-Partial-Order.chain ( $\leq$ ) A
  A  $\neq$  {}  $\forall x \in A$ . set-closure-property S x

```

show *set-closure-property S (Sup-class.Sup A)*

```

proof (rule set-closure-propertyI)

```

```

  fix a b s

```

```

  assume asm: a  $\in$  Sup-class.Sup A s  $\wedge$  b  $\in$  Sup-class.Sup A s

```

```

  then obtain  $\Delta a$   $\Delta b$  where  $\Delta a \in A$   $\Delta b \in A$  a  $\in$   $\Delta a$  s b  $\in$   $\Delta b$  s

```

```

    by auto

```

```

  then show S a b  $\subseteq$  Sup-class.Sup A s

```

```

proof (cases  $\Delta a$  s  $\subseteq$   $\Delta b$  s)

```

```

  case True

```

```

    then have S a b  $\subseteq$   $\Delta b$  s

```

```

    by (metis  $\langle \Delta b \in A \rangle$   $\langle a \in \Delta a s \rangle$   $\langle b \in \Delta b s \rangle$  asm0(3) set-closure-property-def
  subsetD)

```

```

    then show ?thesis

```

```

      using  $\langle \Delta b \in A \rangle$  by auto

```

```

  next

```

```

    case False

```

```

    then have  $\Delta b$  s  $\subseteq$   $\Delta a$  s

```

```

      by (metis  $\langle \Delta a \in A \rangle$   $\langle \Delta b \in A \rangle$  asm0(1) chainD le-funD)

```

```

    then have S a b  $\subseteq$   $\Delta a$  s

```

```

    by (metis  $\langle \Delta a \in A \rangle$   $\langle a \in \Delta a s \rangle$   $\langle b \in \Delta b s \rangle$  asm0(3) subsetD set-closure-property-def)

```

```

    then show ?thesis using ⟨ $\Delta a \in A$ ⟩ by auto
  qed
qed
qed

```

definition *supp* :: ('d, 'c, 'a) interp \Rightarrow bool **where**
supp $\Delta \longleftrightarrow (\forall a b s. a \in \Delta s \wedge b \in \Delta s \longrightarrow (\exists x. a \succeq x \wedge b \succeq x \wedge x \in \Delta s))$

lemma *suppI*:
assumes $\bigwedge a b s. a \in \Delta s \wedge b \in \Delta s \Longrightarrow (\exists x. a \succeq x \wedge b \succeq x \wedge x \in \Delta s)$
shows *supp* Δ
by (*simp add: assms supp-def*)

lemma *supp-admissible*:

```

  ccpo.admissible Sup-class.Sup ( $\leq$ ) supp
proof (rule ccpo.admissibleI)
  fix A :: ('c, 'd, 'a) interp set
  assume asm0: Complete-Partial-Order.chain ( $\leq$ ) A
  A  $\neq \{\}$   $\forall x \in A. \text{supp } x$ 
  show supp (Sup-class.Sup A)
proof (rule suppI)
  fix a b s
  assume asm: a  $\in$  Sup-class.Sup A s  $\wedge$  b  $\in$  Sup-class.Sup A s
  then obtain  $\Delta a \Delta b$  where  $\Delta a \in A \Delta b \in A a \in \Delta a s b \in \Delta b s$ 
  by auto
  then show  $\exists x. a \succeq x \wedge b \succeq x \wedge x \in$  Sup-class.Sup A s
proof (cases  $\Delta a s \subseteq \Delta b s$ )
  case True
  then have a  $\in \Delta b s$ 
  using ⟨a  $\in \Delta a s$ ⟩ by blast
  then obtain x where a  $\succeq$  x b  $\succeq$  x x  $\in \Delta b s$ 
  by (metis ⟨ $\Delta b \in A$ ⟩ ⟨b  $\in \Delta b s$ ⟩ asm0(3) supp-def)
  then show ?thesis
  using ⟨ $\Delta b \in A$ ⟩ by auto
  next
  case False
  then have b  $\in \Delta a s$ 
  by (metis ⟨ $\Delta a \in A$ ⟩ ⟨ $\Delta b \in A$ ⟩ ⟨b  $\in \Delta b s$ ⟩ asm0(1) chainD le-funD subsetD)
  then obtain x where a  $\succeq$  x b  $\succeq$  x x  $\in \Delta a s$ 
  using ⟨ $\Delta a \in A$ ⟩ ⟨a  $\in \Delta a s$ ⟩ asm0(3) supp-def by metis
  then show ?thesis using ⟨ $\Delta a \in A$ ⟩ by auto
  qed
qed
qed

```

lemma *Sup-class.Sup {} = empty-interp* **using** *empty-interp-def*
by *fastforce*

lemma *set-closure-prop-empty-all*:
shows *set-closure-property S empty-interp*
and *set-closure-property S full-interp*
apply (*metis empty-interp-def equals0D set-closure-propertyI*)
by (*simp add: full-interp-def set-closure-propertyI*)

lemma *LFP-preserves-set-closure-property-aux*:
assumes *monotonic f*
and *set-closure-property S empty-interp*
and $\bigwedge \Delta. \text{set-closure-property } S \ \Delta \implies \text{set-closure-property } S \ (f \ \Delta)$
shows *set-closure-property S (ccpo-class.fixp f)*
using *set-closure-property-admissible*
proof (*rule fixp-induct[of set-closure-property S]*)
show *set-closure-property S (Sup-class.Sup {})*
by (*simp add: set-closure-property-def*)
show *monotone (\leq) (\leq) f*
by (*metis (full-types) assms(1) le-fun-def monotoneI monotonic-def smaller-interp-def*)
show $\bigwedge x. \text{set-closure-property } S \ x \implies \text{set-closure-property } S \ (f \ x)$
by (*simp add: assms(3)*)
qed

lemma *GFP-preserves-set-closure-property-aux*:
assumes *order-class.mono f*
and *set-closure-property S full-interp*
and $\bigwedge \Delta. \text{set-closure-property } S \ \Delta \implies \text{set-closure-property } S \ (f \ \Delta)$
shows *set-closure-property S (complete-lattice-class.gfp f)*
using *assms(1)*
proof (*rule gfp-ordinal-induct[of f set-closure-property S]*)
show $\bigwedge Sa. \text{set-closure-property } S \ Sa \implies \text{complete-lattice-class.gfp } f \ \leq \ Sa \implies$
set-closure-property S (f Sa)
using *assms(3)* **by** *blast*
fix *M :: ('c, 'd, 'a) interp set*
assume $\forall Sa \in M. \text{set-closure-property } S \ Sa$
show *set-closure-property S (Inf-class.Inf M)*
proof (*rule set-closure-propertyI*)
fix *a b s*
assume $a \in \text{Inf-class.Inf } M \ s \wedge b \in \text{Inf-class.Inf } M \ s$
then have $\bigwedge \Delta. \Delta \in M \implies a \in \Delta \ s \wedge b \in \Delta \ s$
by *simp*
then have $\bigwedge \Delta. \Delta \in M \implies S \ a \ b \subseteq \Delta \ s$
by (*metis $\langle \forall Sa \in M. \text{set-closure-property } S \ Sa \rangle \text{set-closure-property-def}$*)
show $S \ a \ b \subseteq \text{Inf-class.Inf } M \ s$
by (*simp add: $\langle \bigwedge \Delta. \Delta \in M \implies S \ a \ b \subseteq \Delta \ s \rangle \text{complete-lattice-class.INF-greatest}$*)
qed
qed

5.6 Theorems

5.6.1 Greatest Fixed Point

theorem *GFP-is-FP*:

assumes *monotonic f*

shows $f (GFP f) = GFP f$

proof –

let $?u = GFP f$

have $\bigwedge x. x \in D f \implies \text{smaller-interp } x (f ?u)$

proof –

fix x

assume $x \in D f$

then have $\text{smaller-interp } (f x) (f ?u)$

using *assms monotonic-def smaller-interp-D* **by** *blast*

moreover have $\text{smaller-interp } x (f x)$

using *D-def $\langle x \in D f \rangle$* **by** *fastforce*

ultimately show $\text{smaller-interp } x (f ?u)$

using *smaller-interp-trans* **by** *blast*

qed

then have $?u \in D f$

using *D-def GFP-lub* **by** *blast*

then have $f ?u \in D f$

by (*metis CollectI D-def $\langle \bigwedge x. x \in D f \implies \text{smaller-interp } x (f (GFP f)) \rangle$* *assms monotonic-def*)

then show *?thesis*

by (*simp add: $\langle GFP f \in D f \rangle \langle \bigwedge x. x \in D f \implies \text{smaller-interp } x (f (GFP f)) \rangle$* *smaller-interp-D smaller-interp-antisym*)

qed

theorem *GFP-greatest*:

assumes $f u = u$

shows $\text{smaller-interp } u (GFP f)$

by (*simp add: D-def assms smaller-interp-D smaller-interp-refl*)

lemma *same-GFP*:

assumes *monotonic f*

shows $\text{complete-lattice-class.gfp } f = GFP f$

proof –

have $f (GFP f) = GFP f$

using *GFP-is-FP assms* **by** *blast*

then have $\text{smaller-interp } (GFP f) (\text{complete-lattice-class.gfp } f)$

by (*metis complete-lattice-class.gfp-upperbound le-funD order-class.order.eq-iff smaller-interp-def*)

moreover have $f (\text{complete-lattice-class.gfp } f) = \text{complete-lattice-class.gfp } f$

using *assms gfp-fixpoint mono-same* **by** *blast*

then have $\text{smaller-interp } (\text{complete-lattice-class.gfp } f) (GFP f)$

by (*simp add: GFP-greatest*)

ultimately show *?thesis*
 by *simp*
 qed

5.6.2 Least Fixed Point

theorem *LFP-is-FP*:
 assumes *monotonic f*
 shows $f (LFP f) = LFP f$
proof –
 let $?u = LFP f$
 have $\bigwedge x. x \in DD f \implies \text{smaller-interp } (f ?u) x$
proof –
 fix x
 assume $x \in DD f$
 then have $\text{smaller-interp } (f ?u) (f x)$
 using *assms monotonic-def smaller-interp-DD* by *blast*
 moreover have $\text{smaller-interp } (f x) x$
 using *DD-def $\langle x \in DD f \rangle$* by *fastforce*
 ultimately show $\text{smaller-interp } (f ?u) x$
 using *smaller-interp-trans* by *blast*
 qed
 then have $?u \in DD f$
 using *DD-def LFP-glb* by *blast*
 then have $f ?u \in DD f$
 by (*metis (mono-tags, lifting) CollectI DD-def $\langle \bigwedge x. x \in DD f \implies \text{smaller-interp } (f (LFP f)) x \rangle$ assms monotonic-def*)
 then show *?thesis*
 by (*simp add: $\langle LFP f \in DD f \rangle \langle \bigwedge x. x \in DD f \implies \text{smaller-interp } (f (LFP f)) x \rangle$ smaller-interp-DD smaller-interp-antisym*)
 qed

theorem *LFP-least*:
 assumes $f u = u$
 shows $\text{smaller-interp } (LFP f) u$
 by (*simp add: DD-def assms smaller-interp-DD smaller-interp-refl*)

lemma *same-LFP*:
 assumes *monotonic f*
 shows $\text{complete-lattice-class.lfp } f = LFP f$
proof –
 have $f (LFP f) = LFP f$
 using *LFP-is-FP assms* by *blast*
 then have $\text{smaller-interp } (\text{complete-lattice-class.lfp } f) (LFP f)$
 by (*metis complete-lattice-class.lfp-lowerbound le-funE preorder-class.order-refl smaller-interp-def*)
 moreover have $f (\text{complete-lattice-class.gfp } f) = \text{complete-lattice-class.gfp } f$

```

    using assms gfp-fixpoint mono-same by blast
  then have smaller-interp (LFP f) (complete-lattice-class.lfp f)
    by (meson LFP-least assms lfp-fixpoint mono-same)
  ultimately show ?thesis
    by simp
qed

```

lemma *LFP-same*:

```

  assumes monotonic f
  shows ccpo-class.fixp f = LFP f
proof -
  have f (ccpo-class.fixp f) = ccpo-class.fixp f
    by (metis (mono-tags, lifting) assms fixp-unfold mono-same monotoneI order-class.mono-def)
  then have smaller-interp (LFP f) (ccpo-class.fixp f)
    by (simp add: LFP-least)
  moreover have f (LFP f) = LFP f
    using LFP-is-FP assms by blast
  then have ccpo-class.fixp f ≤ LFP f
    by (metis assms fixp-lowerbound mono-same monotoneI order-class.mono-def preorder-class.order-refl)
  ultimately show ?thesis
    by (metis assms lfp-eq-fixp mono-same same-LFP)
qed

```

The following theorem corresponds to Theorem 5 of the paper [5].

theorem *FP-preserves-set-closure-property*:

```

  assumes monotonic f
  and  $\bigwedge \Delta. \text{set-closure-property } S \ \Delta \implies \text{set-closure-property } S \ (f \ \Delta)$ 
  shows set-closure-property S (GFP f)
  and set-closure-property S (LFP f)
  apply (metis GFP-preserves-set-closure-property-aux assms(1) assms(2) mono-same same-GFP set-closure-prop-empty-all(2))
  by (metis LFP-preserves-set-closure-property-aux LFP-same assms(1) assms(2) set-closure-prop-empty-all(1))

```

end

end

6 Properties of Magic Wands

theory *WandProperties*

imports *Distributivity*

begin

context *logic*

begin

lemma *modus-ponens*:

$Star\ P\ (Wand\ P\ Q),\ \Delta \vdash Q$

proof (*rule entailsI*)

fix $\sigma\ s$

assume $\sigma, s, \Delta \models Star\ P\ (Wand\ P\ Q)$

show $\sigma, s, \Delta \models Q$

using $\langle \sigma, s, \Delta \models Star\ P\ (Wand\ P\ Q) \rangle$ *commutative by force*

qed

lemma *transitivity*:

$Star\ (Wand\ A\ B)\ (Wand\ B\ C),\ \Delta \vdash Wand\ A\ C$

proof (*rule entailsI*)

fix $\sigma\ s$

assume $asm0: \sigma, s, \Delta \models Star\ (Wand\ A\ B)\ (Wand\ B\ C)$

then obtain $ab\ bc$ **where** $Some\ \sigma = ab \oplus bc$ $ab, s, \Delta \models Wand\ A\ B$ $bc, s, \Delta \models$
 $Wand\ B\ C$

by *auto*

show $\sigma, s, \Delta \models Wand\ A\ C$

proof (*rule sat-wand*)

fix $a\ \sigma'$

assume $asm1: a, s, \Delta \models A \wedge Some\ \sigma' = \sigma \oplus a$

then obtain aab **where** $Some\ aab = ab \oplus a$

by (*metis* $\langle Some\ \sigma = ab \oplus bc \rangle$ *asso3 commutative compatible-def option.exhaust-sel*)

then have $Some\ \sigma' = aab \oplus bc$

by (*metis* $\langle Some\ \sigma = ab \oplus bc \rangle$ *asm1 asso1 commutative*)

moreover have $aab, s, \Delta \models B$

using $\langle Some\ aab = ab \oplus a \rangle$ $\langle ab, s, \Delta \models Wand\ A\ B \rangle$ *asm1 by auto*

ultimately show $\sigma', s, \Delta \models C$

using $\langle bc, s, \Delta \models Wand\ B\ C \rangle$ *commutative by auto*

qed

qed

lemma *currying1*:

$Wand\ (Star\ A\ B)\ C,\ \Delta \vdash Wand\ A\ (Wand\ B\ C)$

proof (*rule entailsI*)

fix $\sigma\ s$

assume $asm0: \sigma, s, \Delta \models Wand\ (Star\ A\ B)\ C$

show $\sigma, s, \Delta \models Wand\ A\ (Wand\ B\ C)$

proof (*rule sat-wand*)

fix $a\ \sigma'$

assume $asm1: a, s, \Delta \models A \wedge Some\ \sigma' = \sigma \oplus a$

show $\sigma', s, \Delta \models Wand\ B\ C$

proof (*rule sat-wand*)

fix $b\ \sigma''$

assume $asm2: b, s, \Delta \models B \wedge Some\ \sigma'' = \sigma' \oplus b$

then obtain ab **where** $Some\ ab = a \oplus b$

by (*metis* *asm1 asso2 compatible-def option.collapse*)

then have $ab, s, \Delta \models Star\ A\ B$

```

    using asm1 asm2 by auto
  moreover have Some  $\sigma'' = \sigma \oplus ab$ 
    by (metis  $\langle \text{Some } ab = a \oplus b \rangle$  asm1 asm2 asso1)
  ultimately show  $\sigma'', s, \Delta \models C$ 
    using asm0 sat.simps(3) by blast
qed
qed
qed

```

lemma *currying2*:

```

  Wand A (Wand B C),  $\Delta \vdash$  Wand (Star A B) C
proof (rule entailsI)
  fix  $\sigma$  s
  assume asm0:  $\sigma, s, \Delta \models$  Wand A (Wand B C)
  show  $\sigma, s, \Delta \models$  Wand (Star A B) C
  proof (rule sat-wand)
    fix ab  $\sigma'$ 
    assume asm1:  $ab, s, \Delta \models$  Star A B  $\wedge$  Some  $\sigma' = \sigma \oplus ab$ 
    then obtain a b where Some  $ab = a \oplus b$  a, s,  $\Delta \models$  A b, s,  $\Delta \models$  B
      by auto
    then obtain bc where Some  $bc = \sigma \oplus a$ 
      by (metis asm1 asso3 compatible-def option.exhaust-sel)
    then have bc, s,  $\Delta \models$  Wand B C
      using  $\langle a, s, \Delta \models A \rangle$  asm0 by auto
    moreover have Some  $\sigma' = bc \oplus b$ 
      by (metis  $\langle \text{Some } ab = a \oplus b \rangle$   $\langle \text{Some } bc = \sigma \oplus a \rangle$  asm1 asso1)
    ultimately show  $\sigma', s, \Delta \models C$ 
      using  $\langle b, s, \Delta \models B \rangle$  sat.simps(3) by blast
  qed
qed

```

lemma *distribution*:

```

  Star (Wand A B) C,  $\Delta \vdash$  Wand A (Star B C)
proof (rule entailsI)
  fix  $\sigma$  s
  assume asm0:  $\sigma, s, \Delta \models$  Star (Wand A B) C
  then obtain ab c where Some  $\sigma = ab \oplus c$  ab, s,  $\Delta \models$  Wand A B c, s,  $\Delta \models$  C
    by auto
  show  $\sigma, s, \Delta \models$  Wand A (Star B C)
  proof (rule sat-wand)
    fix a  $\sigma'$ 
    assume asm1:  $a, s, \Delta \models$  A  $\wedge$  Some  $\sigma' = \sigma \oplus a$ 
    then obtain b where Some  $b = ab \oplus a$ 
      by (metis  $\langle \text{Some } \sigma = ab \oplus c \rangle$  asso3 commutative compatible-def option.exhaust-sel)
    then have b, s,  $\Delta \models$  B
      using  $\langle ab, s, \Delta \models$  Wand A B  $\rangle$  asm1 by force
    moreover have Some  $\sigma' = b \oplus c$ 
      by (metis  $\langle \text{Some } \sigma = ab \oplus c \rangle$   $\langle \text{Some } b = ab \oplus a \rangle$  asm1 asso1 commutative)
    ultimately show  $\sigma', s, \Delta \models$  Star B C
  qed

```

```

    using ⟨c, s, Δ ⊨ C⟩ sat.simps(2) by blast
  qed
qed

lemma adjunct1:
  assumes A, Δ ⊢ Wand B C
  shows Star A B, Δ ⊢ C
proof (rule entailsI)
  fix σ s
  assume σ, s, Δ ⊨ Star A B
  then show σ, s, Δ ⊨ C
    using assms entails-def by force
qed

lemma adjunct2:
  assumes Star A B, Δ ⊢ C
  shows A, Δ ⊢ Wand B C
proof (rule entailsI)
  fix σ s
  assume σ, s, Δ ⊨ A
  then show σ, s, Δ ⊨ Wand B C
    by (meson assms entails-def sat.simps(2) sat-wand)
qed

end

end

```

7 Fractional Predicates and Magic Wands in Automatic Separation Logic Verifiers

This section corresponds to Section 5 of the paper [5].

```

theory AutomaticVerifiers
  imports FixedPoint WandProperties
begin

context logic
begin

```

7.1 Syntactic multiplication

The following definition corresponds to Figure 6 of the paper [5].

```

fun syn-mult :: 'b ⇒ ('a, 'b, 'c, 'd) assertion ⇒ ('a, 'b, 'c, 'd) assertion where
  syn-mult π (Star A B) = Star (syn-mult π A) (syn-mult π B)
| syn-mult π (Wand A B) = Wand (syn-mult π A) (syn-mult π B)
| syn-mult π (Or A B) = Or (syn-mult π A) (syn-mult π B)

```

```

| syn-mult  $\pi$  (And  $A B$ ) = And (syn-mult  $\pi A$ ) (syn-mult  $\pi B$ )
| syn-mult  $\pi$  (Imp  $A B$ ) = Imp (syn-mult  $\pi A$ ) (syn-mult  $\pi B$ )
| syn-mult  $\pi$  (Mult  $\alpha A$ ) = syn-mult (smult  $\alpha \pi$ )  $A$ 
| syn-mult  $\pi$  (Exists  $x A$ ) = Exists  $x$  (syn-mult  $\pi A$ )
| syn-mult  $\pi$  (Forall  $x A$ ) = Forall  $x$  (syn-mult  $\pi A$ )
| syn-mult  $\pi$  (Wildcard  $A$ ) = Wildcard  $A$ 
| syn-mult  $\pi A$  = Mult  $\pi A$ 

```

definition *div-state* **where**

```

\pi \sigma = (SOME  $r$ .  $\sigma = \pi \odot r$ )

```

lemma *div-state-ok*:

```

 $\sigma = \pi \odot$  (div-state  $\pi \sigma$ )

```

```

by (metis (mono-tags) div-state-def someI-ex unique-inv)

```

The following theorem corresponds to Theorem 6 of the paper [5].

theorem *syn-sen-mult-same*:

```

 $\sigma, s, \Delta \models$  syn-mult  $\pi A \longleftrightarrow \sigma, s, \Delta \models$  Mult  $\pi A$ 

```

proof (*induct* A *arbitrary*: $\sigma \pi s$)

```

case (Exists  $x A$ )

```

```

show ?case (is ? $A \longleftrightarrow ?B$ )

```

```

proof

```

```

  show ? $B \implies ?A$ 

```

```

    using Exists.hyps by auto

```

```

  show ? $A \implies ?B$ 

```

```

    using Exists.hyps by fastforce

```

```

qed

```

```

next

```

```

case (Forall  $x A$ )

```

```

then show ?case

```

```

  by (metis dot-forall1 dot-forall2 entails-def sat.simps(9) syn-mult.simps(8))

```

```

next

```

```

case (Star  $A B$ )

```

```

show ?case (is ? $P \longleftrightarrow ?Q$ )

```

```

proof

```

```

  show ? $P \implies ?Q$ 

```

```

  proof –

```

```

    assume ? $P$ 

```

```

    then obtain  $a b$  where  $a, s, \Delta \models$  syn-mult  $\pi A$   $b, s, \Delta \models$  syn-mult  $\pi B$ 

```

```

    Some  $\sigma = a \oplus b$  by auto

```

```

    then obtain  $a, s, \Delta \models$  Mult  $\pi A$   $b, s, \Delta \models$  Mult  $\pi B$ 

```

```

      using Star.hyps(1) Star.hyps(2) Star.prems by blast

```

```

    then show ? $Q$ 

```

```

      by (meson  $\langle$ Some  $\sigma = a \oplus b \rangle$  dot-star2 entails-def sat.simps(2))

```

```

  qed

```

```

  assume ? $Q$ 

```

```

  then obtain  $a b$  where  $a, s, \Delta \models$  Mult  $\pi A$   $b, s, \Delta \models$  Mult  $\pi B$  Some  $\sigma = a$ 

```

```

 $\oplus b$ 

```

```

  by (meson dot-star1 entails-def sat.simps(2))

```

```

    then show ?P
      using Star.hyps(1) Star.hyps(2) Star.prem by force
  qed
next
case (Mult p A)
show ?case (is ?P  $\longleftrightarrow$  ?Q)
proof
  show ?P  $\implies$  ?Q
  proof -
    assume ?P
    then have  $\sigma, s, \Delta \models \text{syn-mult } (\text{smult } p \ \pi) \ A$  by auto
    then have  $\sigma, s, \Delta \models \text{Mult } (\text{smult } p \ \pi) \ A$ 
      using Mult.hyps by blast
    then show ?Q
      by (metis dot-mult2 logic.entails-def logic-axioms smult-comm)
  qed
  assume ?Q
  then obtain a where  $a, s, \Delta \models A \ \sigma = \pi \odot (p \odot a)$  by auto
  then show ?P
    using Mult.hyps double-mult smult-comm by auto
  qed
next
case (Wand A B)
show ?case (is ?P  $\longleftrightarrow$  ?Q)
proof
  show ?P  $\implies$  ?Q
  proof -
    assume  $\sigma, s, \Delta \models \text{syn-mult } \pi \ (\text{Wand } A \ B)$ 
    then have  $\sigma, s, \Delta \models \text{Wand } (\text{syn-mult } \pi \ A) \ (\text{syn-mult } \pi \ B)$ 
      by auto
    moreover have  $\text{div-state } \pi \ \sigma, s, \Delta \models \text{Wand } A \ B$ 
    proof (rule sat-wand)
      fix a b
      assume  $a, s, \Delta \models A \wedge \text{Some } b = \text{div-state } \pi \ \sigma \oplus a$ 
      then have  $\text{Some } (\pi \odot b) = \sigma \oplus (\pi \odot a)$ 
        using div-state-ok plus-mult by presburger
      moreover have  $\pi \odot a, s, \Delta \models \text{Mult } \pi \ A$ 
        using  $\langle a, s, \Delta \models A \wedge \text{Some } b = \text{div-state } \pi \ \sigma \oplus a \rangle$  by auto
      then have  $\pi \odot a, s, \Delta \models \text{syn-mult } \pi \ A$ 
        using Wand.hyps(1) Wand.prem by blast
      then have  $\pi \odot b, s, \Delta \models \text{syn-mult } \pi \ B$ 
        using  $\langle \sigma, s, \Delta \models \text{Wand } (\text{syn-mult } \pi \ A) \ (\text{syn-mult } \pi \ B) \rangle$  calculation by
    auto
  ultimately show  $b, s, \Delta \models B$ 
    by (metis Wand.hyps(2) Wand.prem can-divide sat.simps(1))
  qed
  then show  $\sigma, s, \Delta \models \text{Mult } \pi \ (\text{Wand } A \ B)$ 
    by (metis div-state-ok sat.simps(1))
  qed
qed

```

```

assume  $\sigma, s, \Delta \models \text{Mult } \pi (Wand A B)$ 
then have  $\text{div-state } \pi \sigma, s, \Delta \models Wand A B$ 
  by (metis div-state-ok can-divide sat.simps(1))
have  $\sigma, s, \Delta \models Wand (\text{syn-mult } \pi A) (\text{syn-mult } \pi B)$ 
proof (rule sat-wand)
  fix  $a b$  assume  $a, s, \Delta \models \text{syn-mult } \pi A \wedge \text{Some } b = \sigma \oplus a$ 
  then have  $\text{Some } (\text{div-state } \pi b) = \text{div-state } \pi \sigma \oplus \text{div-state } \pi a$ 
    by (metis div-state-ok plus-mult unique-inv)
  then have  $\text{div-state } \pi b, s, \Delta \models B$ 
    by (metis (no-types, lifting) Wand.hyps(1) ⟨a, s, Δ ⊢ syn-mult π A ∧ Some b = σ ⊕ a⟩ div-state π σ, s, Δ ⊢ Wand A B div-state-ok logic.can-divide logic-axioms sat.simps(1) sat.simps(3))
    then show  $b, s, \Delta \models \text{syn-mult } \pi B$ 
      using Wand.hyps(2) div-state-ok sat.simps(1) by blast
    qed
  then show  $\sigma, s, \Delta \models \text{syn-mult } \pi (Wand A B)$ 
    by simp
  qed
next
case (And A B)
show ?case (is ?P ⟷ ?Q)
proof
  show ?P ⟹ ?Q
  proof –
    assume ?P
    then obtain  $\sigma, s, \Delta \models \text{syn-mult } \pi A \sigma, s, \Delta \models \text{syn-mult } \pi B$ 
      by auto
    then show ?Q
      by (meson And.hyps(1) And.hyps(2) dot-and2 logic.entails-def logic-axioms sat.simps(7))
    qed
  assume ?Q then show ?P
    using And.hyps(1) And.hyps(2) And.prem by auto
  qed
next
case (Imp A B)
show ?case (is ?P ⟷ ?Q)
proof
  show ?P ⟹ ?Q
  by (metis Imp.hyps(1) Imp.hyps(2) sat.simps(1) sat.simps(5) syn-mult.simps(5) unique-inv)
  assume ?Q then show ?P
    by (metis Imp.hyps(1) Imp.hyps(2) Imp.prem can-divide sat.simps(1) sat.simps(5) syn-mult.simps(5))
  qed
next
case (Wildcard A)
then show ?case
  by (metis DotWild entails-def equivalent-def syn-mult.simps(9))

```

qed (auto)

7.2 Monotonicity and fixed point

```
fun pos-neg-rec-call :: bool  $\Rightarrow$  ('a, 'b, 'c, 'd) assertion  $\Rightarrow$  bool where
  pos-neg-rec-call b Pred  $\longleftrightarrow$  b
| pos-neg-rec-call b (Mult - A)  $\longleftrightarrow$  pos-neg-rec-call b A
| pos-neg-rec-call b (Exists - A)  $\longleftrightarrow$  pos-neg-rec-call b A
| pos-neg-rec-call b (Forall - A)  $\longleftrightarrow$  pos-neg-rec-call b A
| pos-neg-rec-call b (Star A B)  $\longleftrightarrow$  pos-neg-rec-call b A  $\wedge$  pos-neg-rec-call b B
| pos-neg-rec-call b (Or A B)  $\longleftrightarrow$  pos-neg-rec-call b A  $\wedge$  pos-neg-rec-call b B
| pos-neg-rec-call b (And A B)  $\longleftrightarrow$  pos-neg-rec-call b A  $\wedge$  pos-neg-rec-call b B
| pos-neg-rec-call b (Wand A B)  $\longleftrightarrow$  pos-neg-rec-call ( $\neg$  b) A  $\wedge$  pos-neg-rec-call b B
| pos-neg-rec-call b (Imp A B)  $\longleftrightarrow$  pos-neg-rec-call ( $\neg$  b) A  $\wedge$  pos-neg-rec-call b B
| pos-neg-rec-call - (Sem -)  $\longleftrightarrow$  True
| pos-neg-rec-call b (Bounded A)  $\longleftrightarrow$  pos-neg-rec-call b A
| pos-neg-rec-call b (Wildcard A)  $\longleftrightarrow$  pos-neg-rec-call b A
```

lemma pos-neg-rec-call-mono:

```
  assumes pos-neg-rec-call b A
  shows (b  $\longrightarrow$  monotonic (applies-eq A))  $\wedge$  ( $\neg$  b  $\longrightarrow$  non-increasing (applies-eq A))
  using assms
proof (induct A arbitrary: b)
  case (Exists x A)
  then show ?case
    by (meson mono-exists non-increasing-exists pos-neg-rec-call.simps(3))
next
  case (Forall x A)
  then show ?case
    by (meson mono-forall non-increasing-forall pos-neg-rec-call.simps(4))
next
  case (Sem x)
  then show ?case
    by (metis applies-eq.simps mem-Collect-eq mono-sem non-increasingI sat.simps(4)
        smaller-interp-def subsetI)
next
  case (Mult x1a A)
  then show ?case
    using mono-mult non-increasing-mult pos-neg-rec-call.simps(2) by blast
next
  case (Star A1 A2)
  then show ?case
    by (metis mono-star non-inc-star pos-neg-rec-call.simps(5))
next
  case (Wand A1 A2)
  then show ?case
```

```

    by (metis mono-wand non-increasing-wand pos-neg-rec-call.simps(8))
next
case (Or A1 A2)
then show ?case
    by (metis mono-or non-increasing-or pos-neg-rec-call.simps(6))
next
case (And A1 A2)
then show ?case
    by (metis mono-and non-increasing-and pos-neg-rec-call.simps(7))
next
case (Imp A1 A2)
then show ?case
    by (metis mono-imp non-increasing-imp pos-neg-rec-call.simps(9))
next
case Pred
then show ?case
    using mono-interp pos-neg-rec-call.simps(1) by blast
next
case (Bounded A)
then show ?case
    using mono-bounded non-increasing-bounded pos-neg-rec-call.simps(11) by blast
next
case (Wildcard A)
then show ?case
    using mono-wild non-increasing-wild pos-neg-rec-call.simps(12) by blast
qed

```

The following theorem corresponds to Theorem 7 of the paper [5].

theorem *exists-lfp-gfp*:

```

assumes pos-neg-rec-call True A
shows  $\sigma, s, LFP(\text{applies-eq } A) \models A \longleftrightarrow \sigma \in LFP(\text{applies-eq } A) s$ 
    and  $\sigma, s, GFP(\text{applies-eq } A) \models A \longleftrightarrow \sigma \in GFP(\text{applies-eq } A) s$ 
apply (metis LFP-is-FP applies-eq.simps assms mem-Collect-eq pos-neg-rec-call-mono)
by (metis GFP-is-FP applies-eq.simps assms mem-Collect-eq pos-neg-rec-call-mono)

```

7.3 Combinability

definition *combinable-sem* :: $((\text{'d} \Rightarrow \text{'c}) \Rightarrow \text{'a} \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**

```

combinable-sem B  $\longleftrightarrow (\forall a b x s \alpha \beta. B s a \wedge B s b \wedge \text{sadd } \alpha \beta = \text{one} \wedge \text{Some } x = \alpha \odot a \oplus \beta \odot b \longrightarrow B s x)$ 

```

fun *wf-assertion* :: $(\text{'a}, \text{'b}, \text{'c}, \text{'d}) \text{ assertion} \Rightarrow \text{bool}$ **where**

```

wf-assertion Pred  $\longleftrightarrow \text{True}$ 
| wf-assertion (Sem B)  $\longleftrightarrow \text{combinable-sem } B$ 
| wf-assertion (Mult - A)  $\longleftrightarrow \text{wf-assertion } A$ 
| wf-assertion (Forall - A)  $\longleftrightarrow \text{wf-assertion } A$ 
| wf-assertion (Exists x A)  $\longleftrightarrow \text{wf-assertion } A \wedge (\forall \Delta. \text{unambiguous } \Delta A x)$ 
| wf-assertion (Star A B)  $\longleftrightarrow \text{wf-assertion } A \wedge \text{wf-assertion } B$ 
| wf-assertion (And A B)  $\longleftrightarrow \text{wf-assertion } A \wedge \text{wf-assertion } B$ 

```

```

| wf-assertion (Wand A B)  $\longleftrightarrow$  wf-assertion B
| wf-assertion (Imp A B)  $\longleftrightarrow$  pure A  $\wedge$  wf-assertion B
| wf-assertion (Wildcard A)  $\longleftrightarrow$  wf-assertion A
| wf-assertion -  $\longleftrightarrow$  False

```

lemma *wf-implies-combinable*:

```

assumes wf-assertion A
and sem-combinable  $\Delta$ 
shows combinable  $\Delta$  A
using assms
proof (induct A)
case (Exists x A)
then show ?case
by (meson combinable-exists wf-assertion.simps(5))
next
case (Forall x A)
then show ?case
by (meson combinable-forall wf-assertion.simps(4))
next
case (Sem B)
show ?case
proof (rule combinableI)
fix a b p q x  $\sigma$  s
assume a, s,  $\Delta \models$  Sem B  $\wedge$  b, s,  $\Delta \models$  Sem B  $\wedge$  Some x = p  $\odot$  a  $\oplus$  q  $\odot$  b  $\wedge$ 
sadd p q = one
then show x, s,  $\Delta \models$  Sem B
by (metis Sem.prem(1) combinable-sem-def sat.simps(4) wf-assertion.simps(2))
qed
next
case (Mult x1a A)
then show ?case
using combinable-mult wf-assertion.simps(3) by blast
next
case (Star A1 A2)
then show ?case
using combinable-star wf-assertion.simps(6) by blast
next
case (Wand A1 A2)
then show ?case
using combinable-wand wf-assertion.simps(8) by blast
next
case (And A1 A2)
then show ?case
using combinable-and by auto
next
case (Imp A1 A2)
then show ?case

```

```

    using combinable-imp by auto
next
case Pred
show ?case
proof (rule combinableI)
  fix a b p q x σ s
  assume a, s, Δ ⊨ Pred ∧ b, s, Δ ⊨ Pred ∧ Some x = p ⊙ a ⊕ q ⊙ b ∧ sadd
p q = one
  then show x, s, Δ ⊨ Pred
    using assms(2) sat.simps(10) sem-combinableE by metis
qed
next
case (Wildcard A)
then show ?case
  using combinable-wildcard wf-assertion.simps(10) by blast
qed (auto)

```

7.4 Theorems

The following two theorems correspond to the rules shown in Section 5.1 of the paper [5].

theorem *apply-wand*:

```

Star (syn-mult π A) (Mult π (Wand A B)), Δ ⊢ syn-mult π B
proof (rule entailsI)
  fix σ s
  assume asm: σ, s, Δ ⊨ Star (syn-mult π A) (Mult π (Wand A B))
  then obtain x y where Some σ = x ⊕ y x, s, Δ ⊨ syn-mult π A y, s, Δ ⊨
Mult π (Wand A B)
  by auto
  then have y, s, Δ ⊨ Wand (syn-mult π A) (syn-mult π B)
  by (metis syn-mult.simps(2) syn-sen-mult-same)
  then show σ, s, Δ ⊨ syn-mult π B
  using ⟨Some σ = x ⊕ y⟩ ⟨x, s, Δ ⊨ syn-mult π A⟩ ⟨y, s, Δ ⊨ Wand (syn-mult
π A) (syn-mult π B)⟩ commutative by auto
qed

```

theorem *package-wand*:

```

assumes Star F (syn-mult π A), Δ ⊢ syn-mult π B
shows F, Δ ⊢ Mult π (Wand A B)
by (metis adjunct2 assms entails-def syn-mult.simps(2) syn-sen-mult-same)

```

The following four theorems correspond to the rules shown in Section 5.2 of the paper [5].

theorem *fold-lfp*:

```

assumes pos-neg-rec-call True A
shows syn-mult π A, LFP (applies-eq A) ⊢ Mult π Pred
by (simp add: assms entails-def exists-lfp-gfp(1) syn-sen-mult-same)

```

theorem *unfold-lfp*:
assumes *pos-neg-rec-call True A*
shows *Mult π Pred, LFP (applies-eq A) \vdash syn-mult π A*
by (*simp add: assms entails-def exists-lfp-gfp(1) syn-sen-mult-same*)

theorem *fold-gfp*:
assumes *pos-neg-rec-call True A*
shows *syn-mult π A, GFP (applies-eq A) \vdash Mult π Pred*
by (*simp add: assms entails-def exists-lfp-gfp(2) syn-sen-mult-same*)

theorem *unfold-gfp*:
assumes *pos-neg-rec-call True A*
shows *Mult π Pred, GFP (applies-eq A) \vdash syn-mult π A*
by (*simp add: assms entails-def exists-lfp-gfp(2) syn-sen-mult-same*)

The following theorems correspond to the rule shown in Section 5.3 of the paper [5].

theorem *wf-assertion-combinable-lfp*:
assumes *wf-assertion A*
and *pos-neg-rec-call True A*
shows *sem-combinable (LFP (applies-eq A))*
proof –
let *?f = $\lambda a b. \{ \sigma \mid \sigma p q. sadd p q = one \wedge Some \sigma = p \odot a \oplus q \odot b \}$*
have *set-closure-property ?f (LFP (applies-eq A))*
proof (*rule FP-preserves-set-closure-property(2)*)
show *monotonic (applies-eq A)*
using *assms(2) pos-neg-rec-call-mono* **by** *blast*
fix $\Delta :: ('d, 'c, 'a) \text{interp}$ **assume** *asm0: set-closure-property ?f Δ*
then have *sem-combinable Δ*
by (*metis combinable-set-closure*)
then show *set-closure-property ?f (applies-eq A Δ)*
by (*metis assms(1) combinable-set-closure sem-combinable-equiv wf-implies-combinable*)
qed
then show *?thesis* **using** *combinable-set-closure* **by** *metis*
qed

theorem *wf-assertion-combinable-gfp*:
assumes *wf-assertion A*
and *pos-neg-rec-call True A*
shows *sem-combinable (GFP (applies-eq A))*
proof –
let *?f = $\lambda a b. \{ \sigma \mid \sigma p q. sadd p q = one \wedge Some \sigma = p \odot a \oplus q \odot b \}$*
have *set-closure-property ?f (GFP (applies-eq A))*
proof (*rule FP-preserves-set-closure-property(1)*)
show *monotonic (applies-eq A)*
using *assms(2) pos-neg-rec-call-mono* **by** *blast*
fix $\Delta :: ('d, 'c, 'a) \text{interp}$ **assume** *asm0: set-closure-property ?f Δ*
then have *sem-combinable Δ*

```

    by (metis combinable-set-closure)
  then show set-closure-property ?f (applies-eq A  $\Delta$ )
    by (metis assms(1) combinable-set-closure sem-combinable-equiv wf-implies-combinable)
qed
then show ?thesis using combinable-set-closure by metis
qed

```

theorem *wf-combine*:

```

  assumes wf-assertion A
    and pos-neg-rec-call True A
  shows Star (Mult  $\alpha$  Pred) (Mult  $\beta$  Pred), LFP (applies-eq A)  $\vdash$  Mult (sadd  $\alpha$ 
 $\beta$ ) Pred
    and Star (Mult  $\alpha$  Pred) (Mult  $\beta$  Pred), GFP (applies-eq A)  $\vdash$  Mult (sadd  $\alpha$ 
 $\beta$ ) Pred
  apply (metis assms(1) assms(2) logic.combinable-def logic.wf-implies-combinable
logic-axioms wf-assertion.simps(1) wf-assertion-combinable-lfp)
  by (metis assms(1) assms(2) logic.combinable-def logic.wf-implies-combinable
logic-axioms wf-assertion.simps(1) wf-assertion-combinable-gfp)

```

end

end

References

- [1] S. Blom, S. Darabi, M. Huisman, and W. Oortwijn. The VerCors tool set: Verification of parallel and concurrent software. In N. Polikarpova and S. Schneider, editors, *Integrated Formal Methods*, pages 102–110, Cham, 2017. Springer International Publishing.
- [2] R. Bornat, C. Calcagno, P. W. O’Hearn, and M. J. Parkinson. Permission accounting in separation logic. In J. Palsberg and M. Abadi, editors, *Principle of Programming Languages (POPL)*, pages 259–270. ACM, 2005.
- [3] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis (SAS)*, pages 55–72, 2003.
- [4] J. Brotherston, D. Costa, A. Hobor, and J. Wickerson. Reasoning over permissions regions in concurrent separation logic. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification (CAV)*, 2020.
- [5] T. Dardinier, P. Müller, and A. J. Summers. Fractional resources in unbounded separation logic. *Proc. ACM Program. Lang.*, 6(OOPSLA2), oct 2022.
- [6] T. Dardinier, G. Parthasarathy, N. Weeks, P. Müller, and A. J. Summers. Sound automation of magic wands. In S. Shoham and Y. Vizel,

- editors, *Computer Aided Verification*, pages 130–151, Cham, 2022. Springer International Publishing.
- [7] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In M. G. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, editors, *NASA Formal Methods (NFM)*, volume 6617 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2011.
- [8] X.-B. Le and A. Hobor. Logical reasoning for disjoint permissions. In A. Ahmed, editor, *European Symposium on Programming (ESOP)*, 2018.
- [9] K. R. M. Leino, P. Müller, and J. Smans. Verification of concurrent programs with Chalice. In *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 195–222. Springer, 2009.
- [10] P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In B. Jobstmann and K. R. M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 9583, pages 41–62. Springer, 2016.
- [11] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Logic in Computer Science (LICS)*, pages 55–74. IEEE, 2002.