A Modular Formalization of Superposition

Martin Desharnais   Balazs Toth

October 30, 2024

Abstract

Superposition is an efficient proof calculus for reasoning about first-order logic with equality that is implemented in many automatic theorem provers. It works by saturating the given set of clauses and is refutationally complete, meaning that if the set is inconsistent, the saturation will contain a contradiction. In this formalization, we restructured the completeness proof to cleanly separate the ground (i.e., variable-free) and nonground aspects. We relied on the IsaFoR library for first-order terms and on the Isabelle saturation framework. A paper describing this formalization was published at the 15th International Conference on Interactive Theorem Proving (ITP 2024) [1].

Contents

theory *Transitive-Closure-Extra*
imports *Main*
begin

lemma *reflclp-iff*: $\bigwedge R\ x\ y.\ R^{==}\ x\ y \longleftrightarrow R\ x\ y \lor x = y$
by (*metis* (*full-types*) *sup2CI sup2E*)

lemma *reflclp-refl*: $R^{==}\ x\ x$
by *simp*

lemma *transpD-strict-non-strict*:
assumes *transp R*
shows $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R^{==}\ y\ z \Longrightarrow R\ x\ z$
using ‹*transp R*›[*THEN transpD*] **by** *blast*

lemma *transpD-non-strict-strict*:
assumes *transp R*
shows $\bigwedge x\ y\ z.\ R^{==}\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
using ‹*transp R*›[*THEN transpD*] **by** *blast*

lemma *mem-rtrancl-union-iff-mem-rtrancl-lhs*:
assumes $\bigwedge z.\ (x,\ z) \in A^* \Longrightarrow z \notin Domain\ B$
shows $(x,\ y) \in (A \cup B)^* \longleftrightarrow (x,\ y) \in A^*$
using *assms*
by (*meson Domain.DomainI in-rtrancl-UnI rtrancl-Un-separatorE*)

lemma *mem-rtrancl-union-iff-mem-rtrancl-rhs*:
assumes
$\bigwedge z.\ (x,\ z) \in B^* \Longrightarrow z \notin Domain\ A$
shows $(x,\ y) \in (A \cup B)^* \longleftrightarrow (x,\ y) \in B^*$
using *assms*
by (*metis mem-rtrancl-union-iff-mem-rtrancl-lhs sup-commute*)

```
end
theory Abstract-Rewriting-Extra
  imports
    Transitive-Closure-Extra
    Abstract−Rewriting.Abstract-Rewriting
begin

lemma mem-join-union-iff-mem-join-lhs:
  assumes
    ⋀z. (x, z) ∈ A* ⟹ z ∉ Domain B and
    ⋀z. (y, z) ∈ A* ⟹ z ∉ Domain B
  shows (x, y) ∈ (A ∪ B)↓ ⟷ (x, y) ∈ A↓
proof (rule iffI)
  assume (x, y) ∈ (A ∪ B)↓
  then obtain z where
    (x, z) ∈ (A ∪ B)* and (y, z) ∈ (A ∪ B)*
    by auto

  show (x, y) ∈ A↓
  proof (rule joinI)
    from assms(1) show (x, z) ∈ A*
      using ‹(x, z) ∈ (A ∪ B)*› mem-rtrancl-union-iff-mem-rtrancl-lhs[of x A B z]
by simp
  next
    from assms(2) show (y, z) ∈ A*
      using ‹(y, z) ∈ (A ∪ B)*› mem-rtrancl-union-iff-mem-rtrancl-lhs[of y A B z]
by simp
  qed
next
  show (x, y) ∈ A↓ ⟹ (x, y) ∈ (A ∪ B)↓
    by (metis UnCI join-mono subset-Un-eq sup.left-idem)
qed

lemma mem-join-union-iff-mem-join-rhs:
  assumes
    ⋀z. (x, z) ∈ B* ⟹ z ∉ Domain A and
    ⋀z. (y, z) ∈ B* ⟹ z ∉ Domain A
  shows (x, y) ∈ (A ∪ B)↓ ⟷ (x, y) ∈ B↓
  using mem-join-union-iff-mem-join-lhs
  by (metis assms(1) assms(2) sup-commute)

lemma refl-join: refl (r↓)
  by (simp add: joinI-right reflI)

lemma trans-join:
  assumes strongly-norm: SN r and confluent: WCR r
  shows trans (r↓)
proof −
  from confluent strongly-norm have CR r
```

```
    using Newman by metis
  hence r↔* = r↓
    using CR-imp-conversionIff-join by metis
  thus ?thesis
    using conversion-trans by metis
qed

end
theory Term-Rewrite-System
  imports
    Regular-Tree-Relations.Ground-Ctxt
begin

definition compatible-with-gctxt :: 'f gterm rel ⇒ bool where
  compatible-with-gctxt I ⟷ (∀ t t' ctxt. (t, t') ∈ I ⟶ (ctxt⟨t⟩G, ctxt⟨t'⟩G) ∈ I)

lemma compatible-with-gctxtD:
  compatible-with-gctxt I ⟹ (t, t') ∈ I ⟹ (ctxt⟨t⟩G, ctxt⟨t'⟩G) ∈ I
  by (simp add: compatible-with-gctxt-def)

lemma compatible-with-gctxt-converse:
  assumes compatible-with-gctxt I
  shows compatible-with-gctxt (I⁻¹)
  unfolding compatible-with-gctxt-def
proof (intro allI impI)
  fix t t' ctxt
  assume (t, t') ∈ I⁻¹
  thus (ctxt⟨t⟩G, ctxt⟨t'⟩G) ∈ I⁻¹
    by (simp add: assms compatible-with-gctxtD)
qed

lemma compatible-with-gctxt-symcl:
  assumes compatible-with-gctxt I
  shows compatible-with-gctxt (I↔)
  unfolding compatible-with-gctxt-def
proof (intro allI impI)
  fix t t' ctxt
  assume (t, t') ∈ I↔
  thus (ctxt⟨t⟩G, ctxt⟨t'⟩G) ∈ I↔
  proof (induction ctxt arbitrary: t t')
    case GHole
    thus ?case by simp
  next
    case (GMore f ts1 ctxt ts2)
    thus ?case
      using assms[unfolded compatible-with-gctxt-def, rule-format]
      by blast
  qed
qed
```

```
lemma compatible-with-gctxt-rtrancl:
  assumes compatible-with-gctxt I
  shows compatible-with-gctxt (I*)
  unfolding compatible-with-gctxt-def
proof (intro allI impI)
  fix t t′ ctxt
  assume (t, t′) ∈ I*
  thus (ctxt⟨t⟩G, ctxt⟨t′⟩G) ∈ I*
  proof (induction t′ rule: rtrancl-induct)
    case base
    show ?case
      by simp
  next
    case (step y z)
    thus ?case
      using assms[unfolded compatible-with-gctxt-def, rule-format]
      by (meson rtrancl.rtrancl-into-rtrancl)
  qed
qed

lemma compatible-with-gctxt-relcomp:
  assumes compatible-with-gctxt I1 and compatible-with-gctxt I2
  shows compatible-with-gctxt (I1 O I2)
  unfolding compatible-with-gctxt-def
proof (intro allI impI)
  fix t t′′ ctxt
  assume (t, t′′) ∈ I1 O I2
  then obtain t′ where (t, t′) ∈ I1 and (t′, t′′) ∈ I2
    by auto

  have (ctxt⟨t⟩G, ctxt⟨t′⟩G) ∈ I1
    using ‹(t, t′) ∈ I1› assms(1) compatible-with-gctxtD by blast
  moreover have (ctxt⟨t′⟩G, ctxt⟨t′′⟩G) ∈ I2
    using ‹(t′, t′′) ∈ I2› assms(2) compatible-with-gctxtD by blast
  ultimately show (ctxt⟨t⟩G, ctxt⟨t′′⟩G) ∈ I1 O I2
    by auto
qed

lemma compatible-with-gctxt-join:
  assumes compatible-with-gctxt I
  shows compatible-with-gctxt (I↓)
  using assms
 by (simp-all add: join-def compatible-with-gctxt-relcomp compatible-with-gctxt-rtrancl
     compatible-with-gctxt-converse)

lemma compatible-with-gctxt-conversion:
  assumes compatible-with-gctxt I
  shows compatible-with-gctxt (I↔*)
```

by (*simp add*: *assms compatible-with-gctxt-rtrancl compatible-with-gctxt-symcl conversion-def*)

definition *rewrite-inside-gctxt* :: *'f gterm rel* ⇒ *'f gterm rel* **where**
rewrite-inside-gctxt R = {(*ctxt*⟨*t1*⟩$_G$, *ctxt*⟨*t2*⟩$_G$) | *ctxt t1 t2*. (*t1*, *t2*) ∈ *R*}

lemma *mem-rewrite-inside-gctxt-if-mem-rewrite-rules*[*intro*]:
(*l*, *r*) ∈ *R* ⟹ (*l*, *r*) ∈ *rewrite-inside-gctxt R*
by (*metis* (*mono-tags*, *lifting*) *CollectI gctxt-apply-term.simps*(*1*) *rewrite-inside-gctxt-def*)

lemma *ctxt-mem-rewrite-inside-gctxt-if-mem-rewrite-rules*[*intro*]:
(*l*, *r*) ∈ *R* ⟹ (*ctxt*⟨*l*⟩$_G$, *ctxt*⟨*r*⟩$_G$) ∈ *rewrite-inside-gctxt R*
by (*auto simp*: *rewrite-inside-gctxt-def*)

lemma *rewrite-inside-gctxt-mono*: *R* ⊆ *S* ⟹ *rewrite-inside-gctxt R* ⊆ *rewrite-inside-gctxt S*
by (*auto simp add*: *rewrite-inside-gctxt-def*)

lemma *rewrite-inside-gctxt-union*:
rewrite-inside-gctxt (*R* ∪ *S*) = *rewrite-inside-gctxt R* ∪ *rewrite-inside-gctxt S*
by (*auto simp add*: *rewrite-inside-gctxt-def*)

lemma *rewrite-inside-gctxt-insert*:
rewrite-inside-gctxt (*insert r R*) = *rewrite-inside-gctxt* {*r*} ∪ *rewrite-inside-gctxt R*
using *rewrite-inside-gctxt-union*[*of* {*r*} *R*, *simplified*] **.**

lemma *converse-rewrite-steps*: (*rewrite-inside-gctxt R*)$^{-1}$ = *rewrite-inside-gctxt* (R^{-1})
by (*auto simp*: *rewrite-inside-gctxt-def*)

lemma *rhs-lt-lhs-if-rule-in-rewrite-inside-gctxt*:
fixes *less-trm* :: *'f gterm* ⇒ *'f gterm* ⇒ *bool* (**infix** $\prec_t$ *50*)
assumes
rule-in: (*t1*, *t2*) ∈ *rewrite-inside-gctxt R* **and**
ball-R-rhs-lt-lhs: ⋀*t1 t2*. (*t1*, *t2*) ∈ *R* ⟹ *t2* $\prec_t$ *t1* **and**
compatible-with-gctxt: ⋀*t1 t2 ctxt*. *t2* $\prec_t$ *t1* ⟹ *ctxt*⟨*t2*⟩$_G$ $\prec_t$ *ctxt*⟨*t1*⟩$_G$
shows *t2* $\prec_t$ *t1*
proof −
from *rule-in* **obtain** *t1'* *t2'* *ctxt* **where**
(*t1'*, *t2'*) ∈ *R* **and**
t1 = *ctxt*⟨*t1'*⟩$_G$ **and**
t2 = *ctxt*⟨*t2'*⟩$_G$
by (*auto simp*: *rewrite-inside-gctxt-def*)

from *ball-R-rhs-lt-lhs* **have** *t2'* $\prec_t$ *t1'*
using ‹(*t1'*, *t2'*) ∈ *R*› **by** *simp*

with *compatible-with-gctxt* **have** *ctxt*⟨*t2'*⟩$_G$ $\prec_t$ *ctxt*⟨*t1'*⟩$_G$
by *metis*

```
  thus ?thesis
    using ‹t1 = ctxt⟨t1'⟩G› ‹t2 = ctxt⟨t2'⟩G› by metis
qed

lemma mem-rewrite-step-union-NF:
  assumes (t, t') ∈ rewrite-inside-gctxt (R1 ∪ R2)
    t ∈ NF (rewrite-inside-gctxt R2)
  shows (t, t') ∈ rewrite-inside-gctxt R1
  using assms
  unfolding rewrite-inside-gctxt-union
  by blast

lemma predicate-holds-of-mem-rewrite-inside-gctxt:
  assumes rule-in: (t1, t2) ∈ rewrite-inside-gctxt R and
    ball-P: ⋀t1 t2. (t1, t2) ∈ R ⟹ P t1 t2 and
   preservation: ⋀t1 t2 ctxt σ. (t1, t2) ∈ R ⟹ P t1 t2 ⟹ P ctxt⟨t1⟩G ctxt⟨t2⟩G
  shows P t1 t2
proof −
  from rule-in obtain t1' t2' ctxt σ where
    (t1', t2') ∈ R and
    t1 = ctxt⟨t1'⟩G and
    t2 = ctxt⟨t2'⟩G
    by (auto simp: rewrite-inside-gctxt-def)
  thus ?thesis
    using ball-P[OF ‹(t1', t2') ∈ R›]
    using preservation[OF ‹(t1', t2') ∈ R›, of ctxt]
    by simp
qed

lemma compatible-with-gctxt-rewrite-inside-gctxt[simp]: compatible-with-gctxt (rewrite-inside-gctxt
E)
  unfolding compatible-with-gctxt-def rewrite-inside-gctxt-def
  unfolding mem-Collect-eq
  by (metis Pair-inject ctxt-ctxt)

lemma subset-rewrite-inside-gctxt[simp]: E ⊆ rewrite-inside-gctxt E
proof (rule Set.subsetI)
  fix e assume e-in: e ∈ E
  moreover obtain s t where e-def: e = (s, t)
    by fastforce
  show e ∈ rewrite-inside-gctxt E
    unfolding rewrite-inside-gctxt-def
    unfolding mem-Collect-eq
  proof (intro exI conjI)
    show e = (□G⟨s⟩G, □G⟨t⟩G)
      unfolding e-def gctxt-apply-term.simps ..
  next
    show (s, t) ∈ E
```

using *e-in*
unfolding *e-def* **.**
qed
qed

lemma *wf-converse-rewrite-inside-gctxt*:
fixes E :: *'f gterm rel*
assumes
wfP-R: *wfP R* **and**
R-compatible-with-gctxt: $\bigwedge ctxt\ t\ t'.\ R\ t\ t' \Longrightarrow R\ ctxt\langle t\rangle_G\ ctxt\langle t'\rangle_G$ **and**
equations-subset-R: $\bigwedge x\ y.\ (x,\ y) \in E \Longrightarrow R\ y\ x$
shows *wf* $((rewrite\text{-}inside\text{-}gctxt\ E)^{-1})$
proof (*rule wf-subset*)
from *wfP-R* **show** *wf* $\{(x,\ y).\ R\ x\ y\}$
by (*simp add*: *wfP-def*)
next
show $(rewrite\text{-}inside\text{-}gctxt\ E)^{-1} \subseteq \{(x,\ y).\ R\ x\ y\}$
proof (*rule Set.subsetI*)
fix *e* **assume** $e \in (rewrite\text{-}inside\text{-}gctxt\ E)^{-1}$
then obtain *ctxt s t* **where** *e-def*: $e = (ctxt\langle s\rangle_G,\ ctxt\langle t\rangle_G)$ **and** $(t,\ s) \in E$
by (*smt* (*verit*) *Pair-inject converseE mem-Collect-eq rewrite-inside-gctxt-def*)
hence *R s t*
using *equations-subset-R* **by** *simp*
hence $R\ ctxt\langle s\rangle_G\ ctxt\langle t\rangle_G$
using *R-compatible-with-gctxt* **by** *simp*
then show $e \in \{(x,\ y).\ R\ x\ y\}$
by (*simp add*: *e-def*)
qed
qed

end
theory *Ground-Critical-Pairs*
imports *Term-Rewrite-System*
begin

definition *ground-critical-pairs* :: *'f gterm rel* ⇒ *'f gterm rel* **where**
ground-critical-pairs $R = \{(ctxt\langle r_2\rangle_G,\ r_1) \mid ctxt\ l\ r_1\ r_2.\ (ctxt\langle l\rangle_G,\ r_1) \in R \wedge (l,\ r_2) \in R\}$

abbreviation *ground-critical-pair-theorem* :: *'f gterm rel* ⇒ *bool* **where**
ground-critical-pair-theorem (*R* :: *'f gterm rel*) ≡
WCR (*rewrite-inside-gctxt R*) $\longleftrightarrow$ *ground-critical-pairs R* $\subseteq$ $(rewrite\text{-}inside\text{-}gctxt\ R)^{\downarrow}$

end
theory *Multiset-Extra*
imports
HOL−Library.Multiset
HOL−Library.Multiset-Order

```
  Nested-Multisets-Ordinals.Multiset-More
begin

lemma one-le-countE:
  assumes 1 ≤ count M x
  obtains M' where M = add-mset x M'
  using assms by (meson count-greater-eq-one-iff multi-member-split)

lemma two-le-countE:
  assumes 2 ≤ count M x
  obtains M' where M = add-mset x (add-mset x M')
  using assms
 by (metis Suc-1 Suc-eq-plus1-left Suc-leD add.right-neutral count-add-mset multi-member-split
     not-in-iff not-less-eq-eq)

lemma three-le-countE:
  assumes 3 ≤ count M x
  obtains M' where M = add-mset x (add-mset x (add-mset x M'))
  using assms
 by (metis One-nat-def Suc-1 Suc-leD add-le-cancel-left count-add-mset numeral-3-eq-3
plus-1-eq-Suc
     two-le-countE)

lemma one-step-implies-multp_HO-strong:
  fixes A B J K :: - multiset
  defines J ≡ B − A and K ≡ A − B
  assumes J ≠ {#} and ∀ k ∈# K. ∃ x ∈# J. R k x
  shows multp_HO R A B
  unfolding multp_HO-def
proof (intro conjI allI impI)
  show A ≠ B
    using assms by force
next
  show ⋀y. count B y < count A y ⟹ ∃ x. R y x ∧ count A x < count B x
    using assms by (metis in-diff-count)
qed

lemma Uniq-antimono: Q ≤ P ⟹ Uniq Q ≥ Uniq P
  unfolding le-fun-def le-bool-def
  by (rule impI) (simp only: Uniq-I Uniq-D)

lemma Uniq-antimono': (⋀x. Q x ⟹ P x) ⟹ Uniq P ⟹ Uniq Q
  by (fact Uniq-antimono[unfolded le-fun-def le-bool-def, rule-format])

lemma multp-singleton-right[simp]:
  assumes transp R
  shows multp R M {#x#} ⟷ (∀ y ∈# M. R y x)
proof (rule iffI)
  show ∀ y ∈# M. R y x ⟹ multp R M {#x#}
```

using *one-step-implies-multp*[*of* {#*x*#} - *R* {#}, *simplified*] **.**
next
show *multp R M* {#*x*#} ⟹ ∀ *y*∈#*M*. *R y x*
using *multp-implies-one-step*[*OF* ‹*transp R*›]
by (*smt* (*verit*, *del-insts*) *add-0 set-mset-add-mset-insert set-mset-empty single-is-union singletonD*)
qed

lemma *multp-singleton-left*[*simp*]:
assumes *transp R*
shows *multp R* {#*x*#} *M* ⟷ ({#*x*#} ⊂# *M* ∨ (∃ *y* ∈# *M*. *R x y*))
proof (*rule iffI*)
show {#*x*#} ⊂# *M* ∨ (∃ *y*∈#*M*. *R x y*) ⟹ *multp R* {#*x*#} *M*
proof (*elim disjE bexE*)
show {#*x*#} ⊂# *M* ⟹ *multp R* {#*x*#} *M*
by (*simp add*: *subset-implies-multp*)
next
show ⋀*y*. *y* ∈# *M* ⟹ *R x y* ⟹ *multp R* {#*x*#} *M*
using *one-step-implies-multp*[*of M* {#*x*#} *R* {#}, *simplified*] **by** *force*
qed
next
show *multp R* {#*x*#} *M* ⟹ {#*x*#} ⊂# *M* ∨ (∃ *y*∈#*M*. *R x y*)
using *multp-implies-one-step*[*OF* ‹*transp R*›, *of* {#*x*#} *M*]
by (*metis* (*no-types*, *opaque-lifting*) *add-cancel-right-left subset-mset.gr-zeroI subset-mset.less-add-same-cancel2 union-commute union-is-single union-single-eq-member*)
qed

lemma *multp-singleton-singleton*[*simp*]: *transp R* ⟹ *multp R* {#*x*#} {#*y*#} ⟷ *R x y*
using *multp-singleton-right*[*of R* {#*x*#} *y*] **by** *simp*

lemma *multp-subset-supersetI*: *transp R* ⟹ *multp R A B* ⟹ *C* ⊆# *A* ⟹ *B* ⊆# *D* ⟹ *multp R C D*
by (*metis subset-implies-multp subset-mset.antisym-conv2 transpE transp-multp*)

lemma *multp-double-doubleI*:
assumes *transp R multp R A B*
shows *multp R* (*A* + *A*) (*B* + *B*)
using *multp-repeat-mset-repeat-msetI*[*OF* ‹*transp R*› ‹*multp R A B*›, *of 2*]
by (*simp add*: *numeral-Bit0*)

lemma *multp-implies-one-step-strong*:
fixes *A B I J K* :: - *multiset*
assumes *transp R* **and** *asymp R* **and** *multp R A B*
defines *J* ≡ *B* − *A* **and** *K* ≡ *A* − *B*
shows *J* ≠ {#} **and** ∀ *k* ∈# *K*. ∃ *x* ∈# *J*. *R k x*
proof −
from *assms* **have** $multp_{HO}$ *R A B*

by (*simp add*: *multp-eq-multp*$_{HO}$)

thus $J \neq \{\#\}$ **and** $\forall\, k \in\#\ K.\ \exists\, x \in\#\ J.\ R\ k\ x$
using *multp*$_{HO}$*-implies-one-step-strong*[*OF* ‹*multp*$_{HO}$ *R A B*›]
by (*simp-all add*: *J-def K-def*)
qed

lemma *multp-double-doubleD*:
assumes *transp R* **and** *asymp R* **and** *multp R* $(A + A)$ $(B + B)$
shows *multp R A B*
proof −
from *assms* **have**
$B + B - (A + A) \neq \{\#\}$ **and**
$\forall\, k{\in}\#A + A - (B + B).\ \exists\, x{\in}\#B + B - (A + A).\ R\ k\ x$
using *multp-implies-one-step-strong*[*OF assms*] **by** *simp-all*

have *multp R* $(A \cap\#\ B + (A - B))$ $(A \cap\#\ B + (B - A))$
proof (*rule one-step-implies-multp*[*of* $B - A$ $A - B$ R $A \cap\#\ B$])
show $B - A \neq \{\#\}$
using ‹$B + B - (A + A) \neq \{\#\}$›
by (*meson Diff-eq-empty-iff-mset mset-subset-eq-mono-add*)
next
show $\forall\, k{\in}\#A - B.\ \exists\, j{\in}\#B - A.\ R\ k\ j$
proof (*intro ballI*)
fix *x* **assume** $x \in\#\ A - B$
hence $x \in\#\ A + A - (B + B)$
by (*simp add*: *in-diff-count*)
then obtain *y* **where** $y \in\#\ B + B - (A + A)$ **and** $R\ x\ y$
using ‹$\forall\, k{\in}\#A + A - (B + B).\ \exists\, x{\in}\#B + B - (A + A).\ R\ k\ x$› **by** *auto*
then show $\exists\, j{\in}\#B - A.\ R\ x\ j$
by (*auto simp add*: *in-diff-count*)
qed
qed

moreover have $A = A \cap\#\ B + (A - B)$
by (*simp add*: *inter-mset-def*)

moreover have $B = A \cap\#\ B + (B - A)$
by (*metis diff-intersect-right-idem subset-mset.add-diff-inverse subset-mset.inf.cobounded2*)

ultimately show *?thesis*
by *argo*
qed

lemma *multp-double-double*:
transp R $\Longrightarrow$ *asymp R* $\Longrightarrow$ *multp R* $(A + A)$ $(B + B)$ $\longleftrightarrow$ *multp R A B*
using *multp-double-doubleD multp-double-doubleI* **by** *metis*

lemma *multp-doubleton-doubleton*[*simp*]:

transp $R \Longrightarrow$ *asymp* $R \Longrightarrow$ *multp* R {#x, x#} {#y, y#} $\longleftrightarrow R\ x\ y$
using *multp-double-double*[*of* R {#x#} {#y#}, *simplified*] **by** *simp*

lemma *multp-single-doubleI*: $M \neq$ {#} $\Longrightarrow$ *multp* $R\ M\ (M + M)$
using *one-step-implies-multp*[*of* M {#} - M, *simplified*] **by** *simp*

lemma *mult1-implies-one-step-strong*:
assumes *trans* r **and** *asym* r **and** $(A, B) \in$ *mult1* r
shows $B - A \neq$ {#} **and** $\forall k \in\# A - B.\ \exists j \in\# B - A.\ (k, j) \in r$
proof −
from ‹$(A, B) \in$ *mult1* r› **obtain** $b\ B'\ A'$ **where**
B-def: $B =$ *add-mset* $b\ B'$ **and**
A-def: $A = B' + A'$ **and**
$\forall a.\ a \in\# A' \longrightarrow (a, b) \in r$
unfolding *mult1-def* **by** *auto*

have $b \notin\# A'$
by (*meson* ‹$\forall a.\ a \in\# A' \longrightarrow (a, b) \in r$› *assms*(*2*) *asym-onD iso-tuple-UNIV-I*)
then have $b \in\# B - A$
by (*simp add*: *A-def B-def*)
thus $B - A \neq$ {#}
by *auto*

show $\forall k \in\# A - B.\ \exists j \in\# B - A.\ (k, j) \in r$
by (*metis A-def B-def* ‹$\forall a.\ a \in\# A' \longrightarrow (a, b) \in r$› ‹$b \in\# B - A$› ‹$b \notin\# A'$›
add-diff-cancel-left′
add-mset-add-single diff-diff-add-mset diff-single-trivial)
qed

lemma *asymp-multp*:
assumes *asymp* R **and** *transp* R
shows *asymp* (*multp* R)
using *asymp-multp*$_{HO}$[*OF assms*]
unfolding *multp-eq-multp*$_{HO}$[*OF assms*].

lemma *multp-doubleton-singleton*: *transp* $R \Longrightarrow$ *multp* R {# x, x #} {# y #} $\longleftrightarrow R\ x\ y$
by (*cases* $x = y$) *auto*

lemma *image-mset-remove1-mset*:
assumes *inj f*
shows *remove1-mset* ($f\ a$) (*image-mset* $f\ X$) = *image-mset* f (*remove1-mset* a X)
using *image-mset-remove1-mset-if*
unfolding *image-mset-remove1-mset-if inj-image-mem-iff*[*OF assms*, *symmetric*]
by *simp*

lemma *multp*$_{DM}$*-map-strong*:
assumes

f-mono: *monotone-on* (*set-mset* (*M1* + *M2*)) *R S f* **and**
M1-lt-M2: *multp*$_{DM}$ *R M1 M2*
shows *multp*$_{DM}$ *S* (*image-mset f M1*) (*image-mset f M2*)
proof −
obtain *Y X* **where**
Y ≠ {#} **and** *Y* ⊆# *M2* **and** *M1-eq*: *M1* = *M2* − *Y* + *X* **and**
ex-y: ∀ *x*. *x* ∈# *X* ⟶ (∃ *y*. *y* ∈# *Y* ∧ *R x y*)
using *M1-lt-M2*[*unfolded multp*$_{DM}$*-def Let-def mset-map*] **by** *blast*

let *?fY* = *image-mset f Y*
let *?fX* = *image-mset f X*

show *?thesis*
unfolding *multp*$_{DM}$*-def*
proof (*intro exI conjI*)
show *image-mset f Y* ≠ {#}
using ‹*Y* ≠ {#}› **unfolding** *image-mset-is-empty-iff* **.**
next
show *image-mset f Y* ⊆# *image-mset f M2*
using ‹*Y* ⊆# *M2*› *image-mset-subseteq-mono* **by** *metis*
next
show *image-mset f M1* = *image-mset f M2* − *?fY* + *?fX*
using *M1-eq*[*THEN arg-cong*, *of image-mset f*] ‹*Y* ⊆# *M2*›
by (*metis image-mset-Diff image-mset-union*)
next
obtain *g* **where** *y*: ∀ *x*. *x* ∈# *X* ⟶ *g x* ∈# *Y* ∧ *R x* (*g x*)
using *ex-y* **by** *moura*

show ∀ *fx*. *fx* ∈# *?fX* ⟶ (∃ *fy*. *fy* ∈# *?fY* ∧ *S fx fy*)
proof (*intro allI impI*)
fix *x'* **assume** *x'* ∈# *?fX*
then obtain *x* **where** *x'*: *x'* = *f x* **and** *x-in*: *x* ∈# *X*
by *auto*
hence *y-in*: *g x* ∈# *Y* **and** *y-gt*: *R x* (*g x*)
using *y*[*rule-format*, *OF x-in*] **by** *blast+*

moreover have *X* ⊆# *M1*
using *M1-eq* **by** *simp*

ultimately have *f* (*g x*) ∈# *?fY* ∧ *S* (*f x*)(*f* (*g x*))
using *f-mono*[*THEN monotone-onD*, *of x g x*] ‹*Y* ⊆# *M2*› ‹*X* ⊆# *M1*›
x-in
by (*metis imageI in-image-mset mset-subset-eqD union-iff*)
thus ∃ *fy*. *fy* ∈# *?fY* ∧ *S x' fy*
unfolding *x'* **by** *auto*
qed
qed
qed

```
lemma multp-map-strong:
  assumes
    transp: transp R and
    f-mono: monotone-on (set-mset (M1 + M2)) R S f and
    M1-lt-M2: multp R M1 M2
  shows multp S (image-mset f M1) (image-mset f M2)
  using monotone-on-multp-multp-image-mset[THEN monotone-onD, OF f-mono
transp - - M1-lt-M2]
  by simp

lemma multp_HO-add-mset:
  assumes asymp R transp R R x y multp_HO R X Y
  shows multp_HO R (add-mset x X) (add-mset y Y)
  unfolding multp_HO-def
proof(intro allI conjI impI)
  show add-mset x X ≠ add-mset y Y
    using assms(1, 3, 4)
    unfolding multp_HO-def
    by (metis asympD count-add-mset lessI less-not-refl)
next
  fix x'
  assume count-x': count (add-mset y Y) x' < count (add-mset x X) x'
  show ∃y'. R x' y' ∧ count (add-mset x X) y' < count (add-mset y Y) y'
  proof(cases x' = x)
      case True
      then show ?thesis
        using assms
        unfolding multp_HO-def
       by (metis count-add-mset irreflpD irreflp-on-if-asymp-on not-less-eq transpE)
    next
      case x'-neq-x: False
      show ?thesis
      proof(cases y = x')
        case True
        then show ?thesis
          using assms(1, 3, 4) count-x' x'-neq-x
          unfolding multp_HO-def count-add-mset
          by (smt (verit) Suc-lessD asympD)
      next
        case False
        then show ?thesis
          using assms count-x' x'-neq-x
          unfolding multp_HO-def count-add-mset
         by (smt (verit, del-insts) irreflpD irreflp-on-if-asymp-on not-less-eq transpE)
      qed
    qed
qed
```

lemma *multp-add-mset*:
  assumes *asymp R transp R R x y multp R X Y*
  shows *multp R* (*add-mset x X*) (*add-mset y Y*)
  using $multp_{HO}$*-add-mset*[*OF assms*(*1−3*)] *assms*(*4*)
  unfolding *multp-eq-*$multp_{HO}$[*OF assms*(*1*, *2*)]
  by *simp*

lemma *multp-add-mset′*:
  assumes *R x y*
  shows *multp R* (*add-mset x X*) (*add-mset y X*)
  using *assms*
  by (*metis add-mset-add-single empty-iff insert-iff one-step-implies-multp set-mset-add-mset-insert*

    set-mset-empty)

lemma *multp-add-mset-reflclp*:
  assumes *asymp R transp R R x y* $(multp\ R)^{==}$ *X Y*
  shows *multp R* (*add-mset x X*) (*add-mset y Y*)
  using
    assms(*4*)
    multp-add-mset′[*of R*, *OF assms*(*3*)]
    multp-add-mset[*OF assms*(*1−3*)]
  by *blast*

lemma *multp-add-same*:
  assumes *asymp R transp R multp R X Y*
  shows *multp R* (*add-mset x X*) (*add-mset x Y*)
  by (*meson assms asymp-on-subset irreflp-on-if-asymp-on multp-cancel-add-mset*
top-greatest)

end
theory *Uprod-Extra*
  imports
    HOL−Library.Multiset
    HOL−Library.Uprod
begin

abbreviation *upair* **where**
  upair ≡ λ(*x*, *y*). *Upair x y*

lemma *Upair-sym*: *Upair x y* = *Upair y x*
  by (*metis Upair-inject*)

lemma *ex-ordered-Upair*:
  assumes *tot*: *totalp-on* (*set-uprod p*) *R*
  shows ∃ *x y*. *p* = *Upair x y* ∧ $R^{==}$ *x y*
proof −
  obtain *x y* **where** *p* = *Upair x y*
    by (*metis uprod-exhaust*)

show *?thesis*
proof (*cases* $R^{==}$ *x y*)
case *True*
show *?thesis*
proof (*intro exI conjI*)
show *p* = *Upair x y*
using ‹*p* = *Upair x y*› **.**
next
show $R^{==}$ *x y*
using *True* **by** *simp*
qed
next
case *False*
then show *?thesis*
proof (*intro exI conjI*)
show *p* = *Upair y x*
using ‹*p* = *Upair x y*› **by** *simp*
next
from *tot* **have** *R y x*
using *False*
by (*simp add*: ‹*p* = *Upair x y*› *totalp-on-def*)
thus $R^{==}$ *y x*
by *simp*
qed
qed
qed

definition *mset-uprod* :: *'a uprod* ⇒ *'a multiset* **where**
mset-uprod = *case-uprod* (*Abs-commute* (λ*x y*. {#*x*, *y*#}))

lemma *Abs-commute-inverse-mset*[*simp*]:
apply-commute (*Abs-commute* (λ*x y*. {#*x*, *y*#})) = (λ*x y*. {#*x*, *y*#})
by (*simp add*: *Abs-commute-inverse*)

lemma *set-mset-mset-uprod*[*simp*]: *set-mset* (*mset-uprod up*) = *set-uprod up*
by (*simp add*: *mset-uprod-def case-uprod.rep-eq set-uprod.rep-eq case-prod-beta*)

lemma *mset-uprod-Upair*[*simp*]: *mset-uprod* (*Upair x y*) = {#*x*, *y*#}
by (*simp add*: *mset-uprod-def*)

lemma *map-uprod-inverse*: (⋀*x*. *f* (*g x*) = *x*) ⟹ (⋀*y*. *map-uprod f* (*map-uprod g y*) = *y*)
by (*simp add*: *uprod.map-comp uprod.map-ident-strong*)

lemma *mset-uprod-image-mset*: *mset-uprod* (*map-uprod f p*) = *image-mset f* (*mset-uprod p*)
proof−
obtain *x y* **where** [*simp*]: *p* = *Upair x y*

using *uprod-exhaust* **by** *blast*

have *mset-uprod* (*map-uprod f p*) = {# *f x*, *f y* #}
by *simp*

then show *mset-uprod* (*map-uprod f p*) = *image-mset f* (*mset-uprod p*)
by *simp*
qed

end
theory *HOL-Extra*
imports *Main*
begin

lemmas *UniqI* = *Uniq-I*

lemma *Uniq-prodI*:
assumes $\bigwedge$*x1 y1 x2 y2*. *P x1 y1* $\Longrightarrow$ *P x2 y2* $\Longrightarrow$ (*x1*, *y1*) = (*x2*, *y2*)
shows $\exists_{\leq 1}$(*x*, *y*). *P x y*
using *assms*
by (*metis UniqI case-prodE*)

lemma *Uniq-implies-ex1*: $\exists_{\leq 1}$*x*. *P x* $\Longrightarrow$ *P y* $\Longrightarrow$ $\exists$!*x*. *P x*
by (*iprover intro*: *ex1I dest*: *Uniq-D*)

lemma *Uniq-antimono*: $Q \leq P \Longrightarrow$ *Uniq Q* $\geq$ *Uniq P*
unfolding *le-fun-def le-bool-def*
by (*rule impI*) (*simp only*: *Uniq-I Uniq-D*)

lemma *Uniq-antimono'*: ($\bigwedge$*x*. *Q x* $\Longrightarrow$ *P x*) $\Longrightarrow$ *Uniq P* $\Longrightarrow$ *Uniq Q*
by (*fact Uniq-antimono*[*unfolded le-fun-def le-bool-def*, *rule-format*])

lemma *Collect-eq-if-Uniq*: ($\exists_{\leq 1}$*x*. *P x*) $\Longrightarrow$ {*x*. *P x*} = {} $\vee$ ($\exists$ *x*. {*x*. *P x*} = {*x*})
using *Uniq-D* **by** *fastforce*

lemma *Collect-eq-if-Uniq-prod*:
($\exists_{\leq 1}$(*x*, *y*). *P x y*) $\Longrightarrow$ {(*x*, *y*). *P x y*} = {} $\vee$ ($\exists$ *x y*. {(*x*, *y*). *P x y*} = {(*x*, *y*)})
using *Collect-eq-if-Uniq* **by** *fastforce*

lemma *Ball-Ex-comm*:
($\forall$ *x* $\in$ *X*. $\exists$ *f*. *P* (*f x*) *x*) $\Longrightarrow$ ($\exists$ *f*. $\forall$ *x* $\in$ *X*. *P* (*f x*) *x*)
($\exists$ *f*. $\forall$ *x* $\in$ *X*. *P* (*f x*) *x*) $\Longrightarrow$ ($\forall$ *x* $\in$ *X*. $\exists$ *f*. *P* (*f x*) *x*)
by *meson+*

lemma *set-map-id*:
assumes *x* $\in$ *set X f x* $\notin$ *set X map f X* = *X*
shows *False*
using *assms*
by(*induction X*) *auto*

end
theory *Relation-Extra*
  imports *HOL.Relation*
begin

lemma *transp-on-empty*[*simp*]: *transp-on* {} *R*
  by (*auto intro*: *transp-onI*)

lemma *asymp-on-empty*[*simp*]: *asymp-on* {} *R*
  by (*auto intro*: *asymp-onI*)

lemma *partition-set-around-element*:
  assumes *tot*: *totalp-on N R* **and** *x-in*: $x \in N$
  shows $N = \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
proof (*intro Set.equalityI Set.subsetI*)
  fix *z* **assume** $z \in N$
  hence $R\ z\ x \vee z = x \vee R\ x\ z$
    using *tot*[*THEN totalp-onD*] *x-in* **by** *auto*
  thus $z \in \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
    using ‹$z \in N$› **by** *auto*
next
  fix *z* **assume** $z \in \{y \in N.\ R\ y\ x\} \cup \{x\} \cup \{y \in N.\ R\ x\ y\}$
  hence $z \in N \vee z = x$
    by *auto*
  thus $z \in N$
    using *x-in* **by** *auto*
qed

end
theory *Clausal-Calculus-Extra*
  imports
    Saturation-Framework-Extensions.Clausal-Calculus
    Uprod-Extra
begin

lemma *map-literal-inverse*:
  $(\bigwedge x.\ f\ (g\ x) = x) \Longrightarrow (\bigwedge literal.\ map\text{-}literal\ f\ (map\text{-}literal\ g\ literal) = literal)$
  by (*simp add*: *literal.map-comp literal.map-ident-strong*)

lemma *map-literal-comp*:
  $map\text{-}literal\ f\ (map\text{-}literal\ g\ literal) = map\text{-}literal\ (\lambda atom.\ f\ (g\ atom))\ literal$
  using *literal.map-comp*
  unfolding *comp-def* **.**

lemma *literals-distinct* [*simp*]: $Neg \neq Pos$ $Pos \neq Neg$
  by(*metis literal.distinct*(*1*))+

primrec *mset-lit* :: *'a uprod literal* ⇒ *'a multiset* **where**

mset-lit (*Pos A*) = *mset-uprod A* |
mset-lit (*Neg A*) = *mset-uprod A* + *mset-uprod A*

lemma *mset-lit-image-mset*: *mset-lit* (*map-literal* (*map-uprod f*) *l*) = *image-mset f* (*mset-lit l*)
by(*induction l*) (*simp-all add*: *mset-uprod-image-mset*)

lemma *uprod-mem-image-iff-prod-mem*[*simp*]:
assumes *sym I*
shows (*Upair t t'*) ∈ (λ(t_1, t_2). *Upair* t_1 t_2) ' *I* ⟷ (*t*, *t'*) ∈ *I*
using ‹*sym I*›[*THEN symD*] **by** *auto*

lemma *true-lit-uprod-iff-true-lit-prod*[*simp*]:
assumes *sym I*
shows
(λ(t_1, t_2). *Upair* t_1 t_2) ' *I* ⊫*l Pos* (*Upair t t'*) ⟷ *I* ⊫*l Pos* (*t*, *t'*)
(λ(t_1, t_2). *Upair* t_1 t_2) ' *I* ⊫*l Neg* (*Upair t t'*) ⟷ *I* ⊫*l Neg* (*t*, *t'*)
unfolding *true-lit-simps uprod-mem-image-iff-prod-mem*[*OF* ‹*sym I*›]
by *simp-all*

end
theory *Ground-Term-Extra*
imports *Regular-Tree-Relations.Ground-Terms*
begin

lemma *gterm-is-fun*: *is-Fun* (*term-of-gterm t*)
by(*cases t*) *simp*

end
theory *Ground-Ctxt-Extra*
imports *Regular-Tree-Relations.Ground-Ctxt*
begin

lemma *le-size-gctxt*: *size t* ≤ *size* ($C\langle t\rangle_G$)
by (*induction C*) *simp-all*

lemma *lt-size-gctxt*: *ctxt* ≠ $\Box_G$ ⟹ *size t* < *size* $ctxt\langle t\rangle_G$
by (*induction ctxt*) *force+*

lemma *gctxt-ident-iff-eq-GHole*[*simp*]: $ctxt\langle t\rangle_G$ = *t* ⟷ *ctxt* = $\Box_G$
proof (*rule iffI*)
assume $ctxt\langle t\rangle_G$ = *t*
hence *size* ($ctxt\langle t\rangle_G$) = *size t*
by *argo*
thus *ctxt* = $\Box_G$
using *lt-size-gctxt*[*of ctxt t*]
by *linarith*
next
show *ctxt* = $\Box_G$ ⟹ $ctxt\langle t\rangle_G$ = *t*

by *simp*
qed

end
theory *Ground-Clause*
imports
Saturation-Framework-Extensions.Clausal-Calculus

Ground-Term-Extra
Ground-Ctxt-Extra
Uprod-Extra
begin

abbreviation *Pos-Upair* (**infix** ≈ *66*) **where**
Pos-Upair x y ≡ *Pos* (*Upair x y*)

abbreviation *Neg-Upair* (**infix** !≈ *66*) **where**
Neg-Upair x y ≡ *Neg* (*Upair x y*)

type-synonym *'f gatom* = *'f gterm uprod*

no-notation *subst-compose* (**infixl** $\circ_s$ *75*)
no-notation *subst-apply-term* (**infixl** · *67*)

end
theory *Selection-Function*
imports
Ground-Clause
begin

locale *select* =
fixes *sel* :: *'a clause* ⇒ *'a clause*
assumes
select-subset: ⋀*C*. *sel C* ⊆# *C* **and**
select-negative-lits: ⋀*C L*. *L* ∈# *sel C* ⟹ *is-neg L*

end
theory *Term-Ordering-Lifting*
imports *Clausal-Calculus-Extra*
begin

lemma *antisymp-on-reflclp-if-asymp-on*:
assumes *asymp-on A R*
shows *antisymp-on A* $R^{==}$
unfolding *antisym-on-reflcl*[*to-pred*]
using *antisymp-on-if-asymp-on*[*OF* ‹*asymp-on A R*›] **.**

lemma *order-reflclp-if-transp-and-asymp*:
  assumes *transp R* **and** *asymp R*
  shows *class.order* $R^{==}$ *R*
proof *unfold-locales*
  show $\bigwedge x\ y.\ R\ x\ y = (R^{==}\ x\ y \wedge \neg\ R^{==}\ y\ x)$
    using ‹*asymp R*› *asympD* **by** *fastforce*
next
  show $\bigwedge x.\ R^{==}\ x\ x$
    by *simp*
next
  show $\bigwedge x\ y\ z.\ R^{==}\ x\ y \Longrightarrow R^{==}\ y\ z \Longrightarrow R^{==}\ x\ z$
    using *transp-on-reflclp*[*OF* ‹*transp R*›, *THEN transpD*] **.**
next
  show $\bigwedge x\ y.\ R^{==}\ x\ y \Longrightarrow R^{==}\ y\ x \Longrightarrow x = y$
    using *antisymp-on-reflclp-if-asymp-on*[*OF* ‹*asymp R*›, *THEN antisympD*] **.**
qed

locale *term-ordering-lifting* =
  fixes
    less-trm :: $'t \Rightarrow 't \Rightarrow$ *bool* (**infix** $\prec_t$ *50*)
  assumes
    transp-less-trm[*intro*]: *transp* $(\prec_t)$ **and**
    asymp-less-trm[*intro*]: *asymp* $(\prec_t)$
begin

definition *less-lit* :: $'t$ *uprod literal* $\Rightarrow$ $'t$ *uprod literal* $\Rightarrow$ *bool* (**infix** $\prec_l$ *50*) **where**
  less-lit L1 L2 ≡ *multp* $(\prec_t)$ (*mset-lit L1*) (*mset-lit L2*)

definition *less-cls* :: $'t$ *uprod clause* $\Rightarrow$ $'t$ *uprod clause* $\Rightarrow$ *bool* (**infix** $\prec_c$ *50*) **where**
  less-cls ≡ *multp* $(\prec_l)$

sublocale *term-order*: *order* $(\prec_t)^{==}$ $(\prec_t)$
  using *order-reflclp-if-transp-and-asymp transp-less-trm asymp-less-trm* **by** *metis*

sublocale *literal-order*: *order* $(\prec_l)^{==}$ $(\prec_l)$
proof (*rule order-reflclp-if-transp-and-asymp*)
  show *transp* $(\prec_l)$
    using *transp-less-trm*
    by (*metis* (*opaque-lifting*) *less-lit-def transp-def transp-multp*)
next
  show *asymp* $(\prec_l)$
  by (*metis asympD asymp-less-trm asymp-multp*$_{HO}$ *asympI less-lit-def multp-eq-multp*$_{HO}$
    transp-less-trm)
qed

sublocale *clause-order*: *order* $(\prec_c)^{==}$ $(\prec_c)$
proof (*rule order-reflclp-if-transp-and-asymp*)
  show *transp* $(\prec_c)$
    by (*simp add*: *less-cls-def transp-multp*)

next
  show *asymp* $(\prec_c)$
    by (*simp add*: *less-cls-def asymp-multp*$_{HO}$ *multp-eq-multp*$_{HO}$)
qed

end

end
theory *Ground-Ordering*
  imports
    Ground-Clause
    Transitive-Closure-Extra
    Clausal-Calculus-Extra
    Min-Max-Least-Greatest.Min-Max-Least-Greatest-Multiset
    Term-Ordering-Lifting
begin

locale *ground-ordering* = *term-ordering-lifting less-trm*
  for
    less-trm :: $'f$ *gterm* $\Rightarrow$ $'f$ *gterm* $\Rightarrow$ *bool* (**infix** $\prec_t$ *50*) +
  assumes
    wfP-less-trm[*intro*]: *wfP* $(\prec_t)$ **and**
    totalp-less-trm[*intro*]: *totalp* $(\prec_t)$ **and**
    less-trm-compatible-with-gctxt[*simp*]: $\bigwedge ctxt\ t\ t'.\ t \prec_t t' \Longrightarrow ctxt\langle t\rangle_G \prec_t ctxt\langle t'\rangle_G$
and
    less-trm-if-subterm[*simp*]: $\bigwedge t\ ctxt.\ ctxt \neq \Box_G \Longrightarrow t \prec_t ctxt\langle t\rangle_G$
begin

abbreviation *lesseq-trm* (**infix** $\preceq_t$ *50*) **where**
  lesseq-trm $\equiv (\prec_t)^{==}$

lemma *lesseq-trm-if-subtermeq*: $t \preceq_t ctxt\langle t\rangle_G$
  using *less-trm-if-subterm*
  by (*metis gctxt-ident-iff-eq-GHole reflclp-iff*)

abbreviation *lesseq-lit* (**infix** $\preceq_l$ *50*) **where**
  lesseq-lit $\equiv (\prec_l)^{==}$

abbreviation *lesseq-cls* (**infix** $\preceq_c$ *50*) **where**
  lesseq-cls $\equiv (\prec_c)^{==}$

lemma *wfP-less-lit*[*simp*]: *wfp* $(\prec_l)$
  unfolding *less-lit-def*
  using *wfP-less-trm wfP-multp wfP-if-convertible-to-wfP* **by** *meson*

lemma *wfP-less-cls*[*simp*]: *wfp* $(\prec_c)$
  using *wfP-less-lit wfP-multp less-cls-def* **by** *metis*

sublocale *term-order*: *linorder lesseq-trm less-trm*
proof *unfold-locales*
  show $\bigwedge x\ y.\ x \preceq_t y \lor y \preceq_t x$
    by (*metis reflclp-iff totalpD totalp-less-trm*)
qed

sublocale *literal-order*: *linorder lesseq-lit less-lit*
proof *unfold-locales*
  have *totalp-on A* $(\prec_l)$ **for** *A*
  proof (*rule totalp-onI*)
    fix *L1 L2* :: *'f gatom literal*
    assume $L1 \neq L2$

    show $L1 \prec_l L2 \lor L2 \prec_l L1$
      unfolding *less-lit-def*
    proof (*rule totalp-multp*[*THEN totalpD*])
      show *totalp* $(\prec_t)$
        using *totalp-less-trm* **.**
    next
      show *transp* $(\prec_t)$
        using *transp-less-trm* **.**
    next
      obtain *x1 y1 x2 y2* :: *'f gterm* **where**
        atm-of L1 = *Upair x1 y1* **and** *atm-of L2* = *Upair x2 y2*
        using *uprod-exhaust* **by** *metis*
      thus *mset-lit L1* ≠ *mset-lit L2*
        using ‹*L1* ≠ *L2*›
        by (*cases L1*; *cases L2*) (*auto simp add*: *add-eq-conv-ex*)
    qed
  qed
  thus $\bigwedge x\ y.\ x \preceq_l y \lor y \preceq_l x$
    by (*metis reflclp-iff totalpD*)
qed

sublocale *clause-order*: *linorder lesseq-cls less-cls*
proof *unfold-locales*
  show $\bigwedge x\ y.\ x \preceq_c y \lor y \preceq_c x$
    unfolding *less-cls-def*
    using *totalp-multp*[*OF literal-order.totalp-on-less literal-order.transp-on-less*]
    by (*metis reflclp-iff totalpD*)
qed

abbreviation *is-maximal-lit* :: *'f gatom literal* ⇒ *'f gatom clause* ⇒ *bool* **where**
  is-maximal-lit L M ≡ *is-maximal-in-mset-wrt* $(\prec_l)$ *M L*

abbreviation *is-strictly-maximal-lit* :: *'f gatom literal* ⇒ *'f gatom clause* ⇒ *bool*
where
  is-strictly-maximal-lit L M ≡ *is-greatest-in-mset-wrt* $(\prec_l)$ *M L*

lemma *less-trm-compatible-with-gctxt'*:
  assumes $ctxt\langle t\rangle_G \prec_t ctxt\langle t'\rangle_G$
  shows $t \prec_t t'$
proof(*rule ccontr*)
  assume $\neg\ t \prec_t t'$
  hence $t' \preceq_t t$
    by *order*

  show *False*
  proof(*cases* $t' = t$)
    case *True*
    then have $ctxt\langle t\rangle_G = ctxt\langle t'\rangle_G$
      by *blast*
    then show *False*
      using *assms* **by** *order*
  next
    case *False*
    then have $t' \prec_t t$
      using ‹$t' \preceq_t t$› **by** *order*

    then have $ctxt\langle t'\rangle_G \prec_t ctxt\langle t\rangle_G$
      using *less-trm-compatible-with-gctxt* **by** *metis*

    then show *?thesis*
      using *assms* **by** *order*
  qed
qed

lemma *less-trm-compatible-with-gctxt-iff*: $ctxt\langle t\rangle_G \prec_t ctxt\langle t'\rangle_G \longleftrightarrow t \prec_t t'$
  using *less-trm-compatible-with-gctxt less-trm-compatible-with-gctxt'*
  by *blast*

lemma *context-less-term-lesseq*:
  assumes
    $\bigwedge t.\ ctxt\langle t\rangle_G \prec_t ctxt'\langle t\rangle_G$
    $t \preceq_t t'$
  shows $ctxt\langle t\rangle_G \prec_t ctxt'\langle t'\rangle_G$
  using *assms less-trm-compatible-with-gctxt*
  by (*metis reflclp-iff term-order.dual-order.strict-trans*)

lemma *context-lesseq-term-less*:
  assumes
    $\bigwedge t.\ ctxt\langle t\rangle_G \preceq_t ctxt'\langle t\rangle_G$
    $t \prec_t t'$
  shows $ctxt\langle t\rangle_G \prec_t ctxt'\langle t'\rangle_G$
  using *assms less-trm-compatible-with-gctxt term-order.dual-order.strict-trans1*
  by *blast*

end

end
theory *Ground-Type-System*
  imports *Ground-Clause*
begin

inductive *welltyped* **for** $\mathcal{F}$ **where**
  GFun: $\mathcal{F}\ f = (\tau s,\ \tau) \Longrightarrow$ *list-all2* (*welltyped* $\mathcal{F}$) *ts* $\tau s \Longrightarrow$ *welltyped* $\mathcal{F}$ (*GFun f ts*) τ

lemma *welltyped-right-unique*: *right-unique* (*welltyped* $\mathcal{F}$)
proof (*rule right-uniqueI*)
  fix $t\ \tau_1\ \tau_2$
  assume *welltyped* $\mathcal{F}\ t\ \tau_1$ **and** *welltyped* $\mathcal{F}\ t\ \tau_2$
  thus $\tau_1 = \tau_2$
    by (*auto elim*!: *welltyped.cases*)
qed

definition $welltyped_a$ **where**
  $welltyped_a\ \mathcal{F}\ A \longleftrightarrow (\exists\,\tau.\ \forall\, t \in$ *set-uprod* $A.$ *welltyped* $\mathcal{F}\ t\ \tau)$

definition $welltyped_l$ **where**
  $welltyped_l\ \mathcal{F}\ L \longleftrightarrow welltyped_a\ \mathcal{F}$ (*atm-of* L)

definition $welltyped_c$ **where**
  $welltyped_c\ \mathcal{F}\ C \longleftrightarrow (\forall\, L \in\#\ C.\ welltyped_l\ \mathcal{F}\ L)$

definition $welltyped_{cs}$ **where**
  $welltyped_{cs}\ \mathcal{F}\ N \longleftrightarrow (\forall\, C \in N.\ welltyped_c\ \mathcal{F}\ C)$

lemma $welltyped_c$*-add-mset*:
  $welltyped_c\ \mathcal{F}$ (*add-mset* $L\ C$) $\longleftrightarrow welltyped_l\ \mathcal{F}\ L \wedge welltyped_c\ \mathcal{F}\ C$
  by (*simp add*: $welltyped_c$*-def*)

lemma $welltyped_c$*-plus*:
  $welltyped_c\ \mathcal{F}\ (C + D) \longleftrightarrow welltyped_c\ \mathcal{F}\ C \wedge welltyped_c\ \mathcal{F}\ D$
  by (*auto simp*: $welltyped_c$*-def*)

lemma *gctxt-apply-term-preserves-typing*:
  assumes
    κ*-type*: *welltyped* $\mathcal{F}\ \kappa\langle t\rangle_G\ \tau_1$ **and**
    t-type: *welltyped* $\mathcal{F}\ t\ \tau_2$ **and**
    t'*-type*: *welltyped* $\mathcal{F}\ t'\ \tau_2$
  shows *welltyped* $\mathcal{F}\ \kappa\langle t'\rangle_G\ \tau_1$
  using κ*-type*
proof (*induction* κ *arbitrary*: τ_1)
  case *GHole*
  then show *?case*
    using *t-type* t'*-type*

```
    using welltyped-right-unique[of ℱ, THEN right-uniqueD]
    by auto
next
  case (GMore f ss1 κ ss2)
  have welltyped ℱ (GFun f (ss1 @ κ⟨t⟩G # ss2)) τ1
    using GMore.prems by simp
  hence welltyped ℱ (GFun f (ss1 @ κ⟨t'⟩G # ss2)) τ1
  proof (cases ℱ GFun f (ss1 @ κ⟨t⟩G # ss2) τ1 rule: welltyped.cases)
    case (GFun τs)
    show ?thesis
    proof (rule welltyped.GFun)
      show ℱ f = (τs, τ1)
        using ‹ℱ f = (τs, τ1)› .
    next
      show list-all2 (welltyped ℱ) (ss1 @ κ⟨t'⟩G # ss2) τs
        using ‹list-all2 (welltyped ℱ) (ss1 @ κ⟨t⟩G # ss2) τs›
        using GMore.IH
        by (smt (verit, del-insts) list-all2-Cons1 list-all2-append1 list-all2-lengthD)
    qed
  qed
  thus ?case
    by simp
qed

end
theory Ground-Superposition
  imports

    Main


    Saturation-Framework.Calculus
    Saturation-Framework-Extensions.Clausal-Calculus
    Abstract−Rewriting.Abstract-Rewriting


    Abstract-Rewriting-Extra
    Ground-Critical-Pairs
    Multiset-Extra
    Term-Rewrite-System
    Transitive-Closure-Extra
    Uprod-Extra
    HOL-Extra
    Relation-Extra
    Clausal-Calculus-Extra
    Selection-Function
    Ground-Ordering
    Ground-Type-System
begin
```

hide-type *Inference-System.inference*
hide-const
Inference-System.Infer
Inference-System.prems-of
Inference-System.concl-of
Inference-System.main-prem-of

no-notation *subst-compose* (**infixl** $\circ_s$ *75*)
no-notation *subst-apply-term* (**infixl** $\cdot$ *67*)

1 Superposition Calculus

locale *ground-superposition-calculus* = *ground-ordering less-trm* + *select select*
for
less-trm :: *'f gterm* ⇒ *'f gterm* ⇒ *bool* (**infix** $\prec_t$ *50*) **and**
select :: *'f gatom clause* ⇒ *'f gatom clause* +
assumes
ground-critical-pair-theorem: $\bigwedge$(*R* :: *'f gterm rel*). *ground-critical-pair-theorem* *R*
begin

1.1 Ground Rules

inductive *ground-superposition* ::
'f gatom clause ⇒ *'f gatom clause* ⇒ *'f gatom clause* ⇒ *bool*
where
ground-superpositionI:
E = *add-mset* L_E E' ⟹
D = *add-mset* L_D D' ⟹
$D \prec_c E$ ⟹
$\mathcal{P} \in \{Pos, Neg\}$ ⟹
$L_E = \mathcal{P}$ (*Upair* $\kappa\langle t\rangle_G$ u) ⟹
$L_D = t \approx t'$ ⟹
$u \prec_t \kappa\langle t\rangle_G$ ⟹
$t' \prec_t t$ ⟹
($\mathcal{P}$ = *Pos* ∧ *select* E = {#} ∧ *is-strictly-maximal-lit* L_E E) ∨
($\mathcal{P}$ = *Neg* ∧ (*select* E = {#} ∧ *is-maximal-lit* L_E E ∨ *is-maximal-lit* L_E (*select* E))) ⟹
select D = {#} ⟹
is-strictly-maximal-lit L_D D ⟹
C = *add-mset* ($\mathcal{P}$ (*Upair* $\kappa\langle t'\rangle_G$ u)) ($E' + D'$) ⟹
ground-superposition D E C

inductive *ground-eq-resolution* ::
'f gatom clause ⇒ *'f gatom clause* ⇒ *bool* **where**
ground-eq-resolutionI:
D = *add-mset* L D' ⟹

$L = Neg\ (Upair\ t\ t) \Longrightarrow$
$select\ D = \{\#\} \wedge is\text{-}maximal\text{-}lit\ L\ D \vee is\text{-}maximal\text{-}lit\ L\ (select\ D) \Longrightarrow$
$C = D' \Longrightarrow$
$ground\text{-}eq\text{-}resolution\ D\ C$

inductive *ground-eq-factoring* ::
$'f\ gatom\ clause \Rightarrow\ 'f\ gatom\ clause \Rightarrow bool$ **where**
ground-eq-factoringI:
$D = add\text{-}mset\ L_1\ (add\text{-}mset\ L_2\ D') \Longrightarrow$
$L_1 = t \approx t' \Longrightarrow$
$L_2 = t \approx t'' \Longrightarrow$
$select\ D = \{\#\} \Longrightarrow$
$is\text{-}maximal\text{-}lit\ L_1\ D \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$C = add\text{-}mset\ (Neg\ (Upair\ t'\ t''))\ (add\text{-}mset\ (t \approx t'')\ D') \Longrightarrow$
$ground\text{-}eq\text{-}factoring\ D\ C$

1.1.1 Alternative Specification of the Superposition Rule

inductive *ground-superposition'* ::
$'f\ gatom\ clause \Rightarrow\ 'f\ gatom\ clause \Rightarrow\ 'f\ gatom\ clause \Rightarrow bool$
where
ground-superposition'I:
$P_1 = add\text{-}mset\ L_1\ P_1' \Longrightarrow$
$P_2 = add\text{-}mset\ L_2\ P_2' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$\mathcal{P} \in \{Pos, Neg\} \Longrightarrow$
$L_1 = \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s') \Longrightarrow$
$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$(\mathcal{P} = Pos \longrightarrow select\ P_1 = \{\#\} \wedge is\text{-}strictly\text{-}maximal\text{-}lit\ L_1\ P_1) \Longrightarrow$
$(\mathcal{P} = Neg \longrightarrow (select\ P_1 = \{\#\} \wedge is\text{-}maximal\text{-}lit\ L_1\ P_1 \vee is\text{-}maximal\text{-}lit\ L_1\ (select\ P_1))) \Longrightarrow$
$select\ P_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal\text{-}lit\ L_2\ P_2 \Longrightarrow$
$C = add\text{-}mset\ (\mathcal{P}\ (Upair\ s\langle t'\rangle_G\ s'))\ (P_1' + P_2') \Longrightarrow$
$ground\text{-}superposition'\ P_2\ P_1\ C$

lemma *ground-superposition'* = *ground-superposition*
proof (*intro ext iffI*)
fix *P1 P2 C*
assume *ground-superposition' P2 P1 C*
thus *ground-superposition P2 P1 C*
proof (*cases P2 P1 C rule*: *ground-superposition'.cases*)
case (*ground-superposition'I* $L_1\ P_1'\ L_2\ P_2'\ \mathcal{P}\ s\ t\ s'\ t'$)
thus *?thesis*
using *ground-superpositionI* **by** *blast*
qed

next
  fix *P1 P2 C*
  assume *ground-superposition P1 P2 C*
  thus *ground-superposition′ P1 P2 C*
  proof (*cases P1 P2 C rule*: *ground-superposition.cases*)
    case (*ground-superpositionI* L_1 P_1' L_2 P_2' $\mathcal{P}$ *s t s′ t′*)
    thus *?thesis*
      using *ground-superposition′I*
      by (*metis literals-distinct*(*2*))
  qed
qed

inductive *ground-pos-superposition* ::
  ′f gatom clause ⇒ *′f gatom clause* ⇒ *′f gatom clause* ⇒ *bool*
where
  ground-pos-superpositionI:
    P_1 = *add-mset* L_1 $P_1' \Longrightarrow$
    P_2 = *add-mset* L_2 $P_2' \Longrightarrow$
    $P_2 \prec_c P_1 \Longrightarrow$
    $L_1 = s\langle t\rangle_G \approx s' \Longrightarrow$
    $L_2 = t \approx t' \Longrightarrow$
    $s' \prec_t s\langle t\rangle_G \Longrightarrow$
    $t' \prec_t t \Longrightarrow$
    select P_1 = {#} $\Longrightarrow$
    is-strictly-maximal-lit L_1 $P_1 \Longrightarrow$
    select P_2 = {#} $\Longrightarrow$
    is-strictly-maximal-lit L_2 $P_2 \Longrightarrow$
    C = *add-mset* $(s\langle t'\rangle_G \approx s')$ $(P_1' + P_2') \Longrightarrow$
    ground-pos-superposition P_2 P_1 C

lemma *ground-superposition-if-ground-pos-superposition*:
  assumes *step*: *ground-pos-superposition* P_2 P_1 C
  shows *ground-superposition* P_2 P_1 C
  using *step*
proof (*cases* P_2 P_1 *C rule*: *ground-pos-superposition.cases*)
  case (*ground-pos-superpositionI* L_1 P_1' L_2 P_2' *s t s′ t′*)
  thus *?thesis*
    using *ground-superpositionI*
    by (*metis insert-iff*)
qed

inductive *ground-neg-superposition* ::
  ′f gatom clause ⇒ *′f gatom clause* ⇒ *′f gatom clause* ⇒ *bool*
where
  ground-neg-superpositionI:
    P_1 = *add-mset* L_1 $P_1' \Longrightarrow$
    P_2 = *add-mset* L_2 $P_2' \Longrightarrow$
    $P_2 \prec_c P_1 \Longrightarrow$
    L_1 = *Neg* (*Upair* $s\langle t\rangle_G$ s') $\Longrightarrow$

$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t \rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
select $P_1 = \{\#\} \wedge$ *is-maximal-lit* L_1 $P_1 \vee$ *is-maximal-lit* L_1 (*select* P_1) $\Longrightarrow$
select $P_2 = \{\#\} \Longrightarrow$
is-strictly-maximal-lit L_2 $P_2 \Longrightarrow$
$C =$ *add-mset* (*Neg* (*Upair* $s\langle t' \rangle_G$ s')) ($P_1' + P_2'$) $\Longrightarrow$
ground-neg-superposition P_2 P_1 C

lemma *ground-superposition-if-ground-neg-superposition*:
  assumes *ground-neg-superposition* P_2 P_1 C
  shows *ground-superposition* P_2 P_1 C
  using *assms*
proof (*cases* P_2 P_1 C *rule*: *ground-neg-superposition.cases*)
  case (*ground-neg-superpositionI* L_1 P_1' L_2 P_2' s t s' t')
  then show *?thesis*
    using *ground-superpositionI*
    by (*metis insert-iff*)
qed

lemma *ground-superposition-iff-pos-or-neg*:
  ground-superposition P_2 P_1 $C \longleftrightarrow$
    ground-pos-superposition P_2 P_1 $C \vee$ *ground-neg-superposition* P_2 P_1 C
proof (*rule iffI*)
  assume *ground-superposition* P_2 P_1 C
  thus *ground-pos-superposition* P_2 P_1 $C \vee$ *ground-neg-superposition* P_2 P_1 C
  proof (*cases* P_2 P_1 C *rule*: *ground-superposition.cases*)
    case (*ground-superpositionI* L_1 P_1' L_2 P_2' $\mathcal{P}$ s t s' t')
    then show *?thesis*
      using *ground-pos-superpositionI*[*of* P_1 L_1 P_1' P_2 L_2 P_2' s t s' t']
      using *ground-neg-superpositionI*[*of* P_1 L_1 P_1' P_2 L_2 P_2' s t s' t']
      by *metis*
  qed
next
  assume *ground-pos-superposition* P_2 P_1 $C \vee$ *ground-neg-superposition* P_2 P_1 C
  thus *ground-superposition* P_2 P_1 C
    using *ground-superposition-if-ground-neg-superposition*
      ground-superposition-if-ground-pos-superposition
    by *metis*
qed

1.2 Ground Layer

definition *G-Inf* :: *'f gatom clause inference set* **where**
  G-Inf =
    {*Infer* [P_2, P_1] C | P_2 P_1 C. *ground-superposition* P_2 P_1 C} ∪
    {*Infer* [P] C | P C. *ground-eq-resolution* P C} ∪
    {*Infer* [P] C | P C. *ground-eq-factoring* P C}

abbreviation *G-Bot* :: *'f gatom clause set* **where**
G-Bot ≡ {{#}}

definition *G-entails* :: *'f gatom clause set* ⇒ *'f gatom clause set* ⇒ *bool* **where**
G-entails N_1 N_2 ⟷ (∀ (*I* :: *'f gterm rel*). *refl I* ⟶ *trans I* ⟶ *sym I* ⟶
compatible-with-gctxt I ⟶ *upair* ' *I* ⊫*s* N_1 ⟶ *upair* ' *I* ⊫*s* N_2)

lemma *ground-superposition-smaller-conclusion*:
assumes
step: *ground-superposition P1 P2 C*
shows $C \prec_c$ *P2*
using *step*
proof (*cases P1 P2 C rule*: *ground-superposition.cases*)
case (*ground-superpositionI* L_1 $P_1{}'$ L_2 $P_2{}'$ $\mathcal{P}$ *s t s' t'*)

have $P_1{}'$ + *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ *s'*)) $P_2{}'$ $\prec_c$ $P_1{}'$ + {#$\mathcal{P}$ (*Upair* $s\langle t\rangle_G$
s')#}
unfolding *less-cls-def*
proof (*intro one-step-implies-multp ballI*)
fix *K* **assume** *K* ∈# *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ *s'*)) $P_2{}'$

moreover have $\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ *s'*) $\prec_l$ $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ *s'*)
proof −
have $s\langle t'\rangle_G \prec_t s\langle t\rangle_G$
using ‹$t' \prec_t t$› *less-trm-compatible-with-gctxt* **by** *simp*
hence *multp* ($\prec_t$) {#$s\langle t'\rangle_G$, *s'*#} {#$s\langle t\rangle_G$, *s'*#}
using *transp-less-trm*
by (*simp add*: *add-mset-commute multp-cancel-add-mset*)

have *?thesis* **if** $\mathcal{P}$ = *Pos*
unfolding *that less-lit-def*
using ‹*multp* ($\prec_t$) {#$s\langle t'\rangle_G$, *s'*#} {#$s\langle t\rangle_G$, *s'*#}› **by** *simp*

moreover have *?thesis* **if** $\mathcal{P}$ = *Neg*
unfolding *that less-lit-def*
using ‹*multp* ($\prec_t$) {#$s\langle t'\rangle_G$, *s'*#} {#$s\langle t\rangle_G$, *s'*#}›
using *multp-double-doubleI* **by** *force*

ultimately show *?thesis*
using ‹$\mathcal{P}$ ∈ {*Pos*, *Neg*}› **by** *auto*
qed

moreover have ∀ *K* ∈# $P_2{}'$. *K* $\prec_l$ $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ *s'*)
proof −
have *is-strictly-maximal-lit* L_2 *P1*
using *ground-superpositionI* **by** *argo*
hence ∀ *K* ∈# $P_2{}'$. ¬ *Pos* (*Upair t t'*) $\prec_l$ *K* ∧ *Pos* (*Upair t t'*) ≠ *K*
unfolding *literal-order.is-greatest-in-mset-iff*
unfolding ‹*P1* = *add-mset* L_2 $P_2{}'$› ‹L_2 = *t* ≈ *t'*›

by *auto*
hence $\forall K \in\# P_2{}'$. $K \prec_l Pos$ (*Upair* t t')
using *literal-order.totalp-on-less*[*THEN totalpD*] **by** *metis*

have *thesis-if-Neg*: *Pos* (*Upair* t t') $\prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
if $\mathcal{P}$ = *Neg*
proof −
have $t \preceq_t s\langle t\rangle_G$
using *lesseq-trm-if-subtermeq* **.**
hence *multp* $(\prec_t)$ $\{\#t, t'\#\}$ $\{\#s\langle t\rangle_G, s', s\langle t\rangle_G, s'\#\}$
unfolding *reflclp-iff*
proof (*elim disjE*)
assume $t \prec_t s\langle t\rangle_G$
moreover hence $t' \prec_t s\langle t\rangle_G$
by (*meson* ‹$t' \prec_t t$› *transpD transp-less-trm*)
ultimately show *?thesis*
by (*auto intro*: *one-step-implies-multp*[*of* - - - {#}, *simplified*])
next
assume $t = s\langle t\rangle_G$
thus *?thesis*
using ‹$t' \prec_t t$›
by (*smt* (*verit*, *ccfv-SIG*) *add.commute add-mset-add-single add-mset-commute*
bex-empty
one-step-implies-multp set-mset-add-mset-insert set-mset-empty
singletonD
union-single-eq-member)
qed
thus *Pos* (*Upair* t t') $\prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
using ‹$\mathcal{P}$ = *Neg*›
by (*simp add*: *less-lit-def*)
qed

have *thesis-if-Pos*: *Pos* (*Upair* t t') $\preceq_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
if $\mathcal{P}$ = *Pos* **and** *is-maximal-lit* L_1 *P2*
proof (*cases s*)
case *GHole*
show *?thesis*
proof (*cases* $t' \preceq_t s'$)
case *True*
hence (*multp* $(\prec_t))^{==}$ $\{\#t, t'\#\}$ $\{\#s\langle t\rangle_G, s'\#\}$
unfolding *GHole*
using *transp-less-trm*
by (*simp add*: *multp-cancel-add-mset*)
thus *?thesis*
unfolding *GHole* ‹$\mathcal{P}$ = *Pos*›
by (*auto simp*: *less-lit-def*)
next
case *False*
hence $s' \prec_t t'$

by *order*
hence *multp* $(\prec_t)$ $\{\#s\langle t\rangle_G,\ s'\#\}$ $\{\#t,\ t'\#\}$
using *transp-less-trm*
by (*simp add*: *GHole multp-cancel-add-mset*)
hence $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s') $\prec_l$ *Pos* (*Upair* t t')
using ‹$\mathcal{P}$ = *Pos*›
by (*simp add*: *less-lit-def*)
moreover have $\forall K \in\# P_1'.\ K \preceq_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
using *that*
unfolding *ground-superpositionI*
unfolding *literal-order.is-maximal-in-mset-iff*
by *auto*
ultimately have $\forall K \in\# P_1'.\ K \preceq_l$ *Pos* (*Upair* t t')
using *literal-order.transp-on-less*
by (*metis* (*no-types*, *lifting*) *reflclp-iff transpD*)
hence *P2* $\prec_c$ *P1*
using ‹$\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s') $\prec_l$ *Pos* (*Upair* t t')›
one-step-implies-multp[*of P1 P2* $(\prec_l)$ $\{\#\}$, *simplified*]
unfolding *ground-superpositionI less-cls-def*
by (*metis* ‹$\forall K{\in}\#P_1'.\ K \preceq_l$ ($\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s'))› *empty-not-add-mset*
insert-iff reflclp-iff
set-mset-add-mset-insert transpD literal-order.transp-on-less)
hence *False*
using ‹*P1* $\prec_c$ *P2*› **by** *order*
thus *?thesis* **..**
qed
next
case (*GMore f ts1 ctxt ts2*)
hence $t \prec_t s\langle t\rangle_G$
using *less-trm-if-subterm*[*of s t*] **by** *simp*
moreover hence $t' \prec_t s\langle t\rangle_G$
using ‹$t' \prec_t t$› **by** *order*
ultimately have *multp* $(\prec_t)$ $\{\#t,\ t'\#\}$ $\{\#s\langle t\rangle_G,\ s'\#\}$
using *one-step-implies-multp*[*of* $\{\#s\langle t\rangle_G,\ s'\#\}$ $\{\#t,\ t'\#\}$ $(\prec_t)$ $\{\#\}$] **by**
simp
hence *Pos* (*Upair* t t') $\prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
using ‹$\mathcal{P}$ = *Pos*›
by (*simp add*: *less-lit-def*)
thus *?thesis*
by *order*
qed

have $\mathcal{P}$ = *Pos* $\vee$ $\mathcal{P}$ = *Neg*
using ‹$\mathcal{P} \in \{$*Pos*, *Neg*$\}$› **by** *simp*
thus *?thesis*
proof (*elim disjE*; *intro ballI*)
fix *K* **assume** $\mathcal{P}$ = *Pos* $K \in\# P_2'$
have $K \prec_l t \approx t'$
using ‹$\forall K{\in}\#P_2'.\ K \prec_l t \approx t'$› ‹$K \in\# P_2'$› **by** *metis*

also have $t \approx t' \preceq_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
proof (*rule thesis-if-Pos*[*OF* ‹$\mathcal{P}$ = *Pos*›])
have *is-strictly-maximal-lit* L_1 *P2*
using ‹$\mathcal{P}$ = *Pos*› *ground-superpositionI literal.simps*(*4*)
by (*metis literal.simps*(*4*))
thus *is-maximal-lit* L_1 *P2*
using *literal-order.is-maximal-in-mset-if-is-greatest-in-mset* **by** *metis*
qed
finally show $K \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s') **.**
next
fix *K* **assume** $\mathcal{P}$ = *Neg* $K \in\#$ $P_2{}'$
have $K \prec_l t \approx t'$
using ‹$\forall K{\in}\#P_2{}'$. $K \prec_l t \approx t'$› ‹$K \in\#$ $P_2{}'$› **by** *metis*
also have $t \approx t' \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')
using *thesis-if-Neg*[*OF* ‹$\mathcal{P}$ = *Neg*›] **.**
finally show $K \prec_l \mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s') **.**
qed
qed

ultimately show $\exists j \in\# \{\#\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')$\#\}$. $K \prec_l j$
by *auto*
qed *simp*

moreover have *C* = *add-mset* ($\mathcal{P}$ (*Upair* $s\langle t'\rangle_G$ s')) ($P_1{}'$ + $P_2{}'$)
unfolding *ground-superpositionI* **..**

moreover have *P2* = $P_1{}'$ + $\{\#\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ s')$\#\}$
unfolding *ground-superpositionI* **by** *simp*

ultimately show *?thesis*
by *simp*
qed

lemma *ground-eq-resolution-smaller-conclusion*:
assumes *step*: *ground-eq-resolution P C*
shows $C \prec_c P$
using *step*
proof (*cases P C rule*: *ground-eq-resolution.cases*)
case (*ground-eq-resolutionI L t*)
then show *?thesis*
using *clause-order.totalp-on-less* **unfolding** *less-cls-def*
by (*metis add.right-neutral add-mset-add-single empty-iff empty-not-add-mset one-step-implies-multp set-mset-empty*)
qed

lemma *ground-eq-factoring-smaller-conclusion*:
assumes *step*: *ground-eq-factoring P C*
shows $C \prec_c P$
using *step*

proof (*cases P C rule*: *ground-eq-factoring.cases*)
  case (*ground-eq-factoringI* L_1 L_2 P' t t' t'')
  have *is-maximal-lit* L_1 P
    using *ground-eq-factoringI* **by** *simp*
  hence $\forall K \in\#$ *add-mset* (*Pos* (*Upair* t t'')) P'. $\neg$ *Pos* (*Upair* t t') $\prec_l$ K
    unfolding *ground-eq-factoringI*
    by (*simp add*: *literal-order.is-maximal-in-mset-iff literal-order.neq-iff*)
  hence $\neg$ *Pos* (*Upair* t t') $\prec_l$ *Pos* (*Upair* t t'')
    by *simp*
  hence *Pos* (*Upair* t t'') $\preceq_l$ *Pos* (*Upair* t t')
    by *order*
  hence $t'' \preceq_t t'$
    unfolding *reflclp-iff*
    using *transp-less-trm*
    by (*auto simp*: *less-lit-def multp-cancel-add-mset*)

  have C = *add-mset* (*Neg* (*Upair* t' t'')) (*add-mset* (*Pos* (*Upair* t t'')) P')
    using *ground-eq-factoringI* **by** *argo*

  moreover have *add-mset* (*Neg* (*Upair* t' t'')) (*add-mset* (*Pos* (*Upair* t t'')) P')
$\prec_c$ P
    unfolding *ground-eq-factoringI less-cls-def*
  proof (*intro one-step-implies-multp*[*of* {#-#} {#-#}, *simplified*])
    have $t'' \prec_t t$
      using ‹$t' \prec_t t$› ‹$t'' \preceq_t t'$› **by** *order*
    hence *multp* ($\prec_t$) {#t', t'', t', t''#} {#t, t'#}
      using *one-step-implies-multp*[*of* - - - {#}, *simplified*]
      by (*metis* ‹$t' \prec_t t$› *diff-empty id-remove-1-mset-iff-notin insert-iff*
        set-mset-add-mset-insert)
    thus *Neg* (*Upair* t' t'') $\prec_l$ *Pos* (*Upair* t t')
      by (*simp add*: *less-lit-def*)
  qed

  ultimately show *?thesis*
    by *argo*
qed

end

sublocale *ground-superposition-calculus* $\subseteq$ *consequence-relation* **where**
  Bot = *G-Bot* **and**
  entails = *G-entails*
proof *unfold-locales*
  show *G-Bot* $\neq$ {}
    by *simp*
next
  show $\bigwedge B\ N.\ B \in$ *G-Bot* $\Longrightarrow$ *G-entails* {B} N
    by (*simp add*: *G-entails-def*)
next

show $\bigwedge$*N2 N1*. *N2* ⊆ *N1* ⟹ *G-entails N1 N2*
by (*auto simp*: *G-entails-def elim*!: *true-clss-mono*[*rotated*])
next
fix *N1 N2* **assume** *ball-G-entails*: ∀ *C* ∈ *N2*. *G-entails N1* {*C*}
show *G-entails N1 N2*
unfolding *G-entails-def*
proof (*intro allI impI*)
fix *I* :: *'f gterm rel*
assume *refl I* **and** *trans I* **and** *sym I* **and** *compatible-with-gctxt I* **and**
(λ(*x*, *y*). *Upair x y*) ‘ *I* ⊫*s N1*
hence ∀ *C* ∈ *N2*. (λ(*x*, *y*). *Upair x y*) ‘ *I* ⊫*s* {*C*}
using *ball-G-entails* **by** (*simp add*: *G-entails-def*)
then show (λ(*x*, *y*). *Upair x y*) ‘ *I* ⊫*s N2*
by (*simp add*: *true-clss-def*)
qed
next
show $\bigwedge$*N1 N2 N3*. *G-entails N1 N2* ⟹ *G-entails N2 N3* ⟹ *G-entails N1 N3*
using *G-entails-def* **by** *simp*
qed

end
theory *Ground-Superposition-Completeness*
imports *Ground-Superposition*
begin

1.3 Redundancy Criterion

sublocale *ground-superposition-calculus* ⊆ *calculus-with-finitary-standard-redundancy*
where
Inf = *G-Inf* **and**
Bot = *G-Bot* **and**
entails = *G-entails* **and**
less = ($\prec_c$)
defines *GRed-I* = *Red-I* **and** *GRed-F* = *Red-F*
proof *unfold-locales*
show *transp* ($\prec_c$)
using *clause-order*.*transp-on-less* **.**
next
show *wfP* ($\prec_c$)
using *wfP-less-cls* **.**
next
show $\bigwedge$ι. ι ∈ *G-Inf* ⟹ *prems-of* ι ≠ []
by (*auto simp*: *G-Inf-def*)
next
fix ι
have *concl-of* ι $\prec_c$ *main-prem-of* ι
if ι*-def*: ι = *Infer* [P_2, P_1] *C* **and**
infer: *ground-superposition* P_2 P_1 *C*
for P_2 P_1 *C*

```
    unfolding ι-def
    using infer
    using ground-superposition-smaller-conclusion
    by simp

  moreover have concl-of ι ≺c main-prem-of ι
    if ι-def: ι = Infer [P] C and
      infer: ground-eq-resolution P C
    for P C
    unfolding ι-def
    using infer
    using ground-eq-resolution-smaller-conclusion
    by simp

  moreover have concl-of ι ≺c main-prem-of ι
    if ι-def: ι = Infer [P] C and
      infer: ground-eq-factoring P C
    for P C
    unfolding ι-def
    using infer
    using ground-eq-factoring-smaller-conclusion
    by simp

  ultimately show ι ∈ G-Inf ⟹ concl-of ι ≺c main-prem-of ι
    unfolding G-Inf-def
    by fast
qed
```

1.4 Mode Construction

```
context ground-superposition-calculus begin

function epsilon :: - ⇒ 'f gatom clause ⇒ 'f gterm rel where
  epsilon N C = {(s, t)| s t C'.
    C ∈ N ∧
    C = add-mset (Pos (Upair s t)) C' ∧
    select C = {#} ∧
    is-strictly-maximal-lit (Pos (Upair s t)) C ∧
    t ≺t s ∧
    (let R_C = (⋃ D ∈ {D ∈ N. D ≺c C}. epsilon {E ∈ N. E ≼c D} D) in
    ¬ upair ` (rewrite-inside-gctxt R_C)↓ ⊫ C ∧
    ¬ upair ` (rewrite-inside-gctxt (insert (s, t) R_C))↓ ⊫ C' ∧
    s ∈ NF (rewrite-inside-gctxt R_C))}
  by auto

termination epsilon
proof (relation {((x1, x2), (y1, y2)). x2 ≺c y2})
  define f :: 'c × 'f gterm uprod literal multiset ⇒ 'f gterm uprod literal multiset
where
```

$f = (\lambda(x1,\ x2).\ x2)$
have $wfp\ (\lambda(x1,\ x2)\ (y1,\ y2).\ x2 \prec_c y2)$
proof $(rule\ wfP\text{-}if\text{-}convertible\text{-}to\text{-}wfP)$
show $\bigwedge x\ y.\ (case\ x\ of\ (x1,\ x2) \Rightarrow \lambda(y1,\ y2).\ x2 \prec_c y2)\ y \Longrightarrow (snd\ x) \prec_c (snd\ y)$
by *auto*
next
show $wfP\ (\prec_c)$
by *simp*
qed
thus $wf\ \{((x1,\ x2),\ (y1,\ y2)).\ x2 \prec_c y2\}$
by (*simp add*: *wfP-def*)
next
show $\bigwedge N\ C\ x\ xa\ xb\ xc\ xd.\ xd \in \{D \in N.\ D \prec_c C\} \Longrightarrow ((\{E \in N.\ E \preceq_c xd\},\ xd),\ N,\ C) \in \{((x1,\ x2),\ y1,\ y2).\ x2 \prec_c y2\}$
by *simp*
qed

declare *epsilon.simps*[*simp del*]

lemma *epsilon-filter-le-conv*: $epsilon\ \{D \in N.\ D \preceq_c C\}\ C = epsilon\ N\ C$
proof (*intro subset-antisym subrelI*)
fix *x y*
assume $(x,\ y) \in epsilon\ \{D \in N.\ D \preceq_c C\}\ C$
then obtain C' **where**
$C \in N$ **and**
$C = add\text{-}mset\ (x \approx y)\ C'$ **and**
$select\ C = \{\#\}$ **and**
$is\text{-}strictly\text{-}maximal\text{-}lit\ (x \approx y)\ C$ **and**
$y \prec_t x$ **and**
$(let\ R_C = \bigcup x{\in}\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\}.\ epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x\ in$
$\neg\ upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ R_C)^{\downarrow} \Vdash C\ \land$
$\neg\ upair\ `\ (rewrite\text{-}inside\text{-}gctxt\ (insert\ (x,\ y)\ R_C))^{\downarrow} \Vdash C'\ \land$
$x \in NF\ (rewrite\text{-}inside\text{-}gctxt\ R_C))$
unfolding *epsilon.simps*[*of - C*] *mem-Collect-eq*
by *auto*

moreover have $(\bigcup x{\in}\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\}.\ epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x) = (\bigcup D{\in}\{D \in N.\ D \prec_c C\}.\ epsilon\ \{E \in N.\ E \preceq_c D\}\ D)$
proof (*rule SUP-cong*)
show $\{D \in N.\ (D \prec_c C \lor D = C) \land D \prec_c C\} = \{D \in N.\ D \prec_c C\}$
by *metis*
next
show $\bigwedge x.\ x \in \{D \in N.\ D \prec_c C\} \Longrightarrow epsilon\ \{E \in N.\ (E \prec_c C \lor E = C) \land E \preceq_c x\}\ x = epsilon\ \{E \in N.\ E \preceq_c x\}\ x$
by (*metis* (*mono-tags*, *lifting*) *clause-order.order.strict-trans1 mem-Collect-eq*)
qed

ultimately show $(x, y) \in$ *epsilon N C*
unfolding *epsilon.simps*[*of - C*] **by** *simp*
next
fix *x y*
assume $(x, y) \in$ *epsilon N C*
then obtain C' **where**
$C \in N$ **and**
$C =$ *add-mset* $(x \approx y)$ C' **and**
select $C = \{\#\}$ **and**
is-strictly-maximal-lit $(x \approx y)$ C **and**
$y \prec_t x$ **and**
(*let* $R_C = \bigcup x \in \{D \in N.\ D \prec_c C\}$. *epsilon* $\{E \in N.\ E \preceq_c x\}$ x *in*
$\neg$ *upair* ' (*rewrite-inside-gctxt* $R_C)^{\downarrow} \models C \wedge$
$\neg$ *upair* ' (*rewrite-inside-gctxt* (*insert* (x, y) $R_C))^{\downarrow} \models C' \wedge$
$x \in NF$ (*rewrite-inside-gctxt* R_C))
unfolding *epsilon.simps*[*of - C*] *mem-Collect-eq*
by *auto*

moreover have ($\bigcup x \in \{D \in N.\ (D \prec_c C \vee D = C) \wedge D \prec_c C\}$. *epsilon* $\{E \in N.\ (E \prec_c C \vee E = C) \wedge E \preceq_c x\}$ x) = ($\bigcup D \in \{D \in N.\ D \prec_c C\}$. *epsilon* $\{E \in N.\ E \preceq_c D\}$ D)
proof (*rule SUP-cong*)
show $\{D \in N.\ (D \prec_c C \vee D = C) \wedge D \prec_c C\} = \{D \in N.\ D \prec_c C\}$
by *metis*
next
show $\bigwedge x.\ x \in \{D \in N.\ D \prec_c C\} \Longrightarrow$ *epsilon* $\{E \in N.\ (E \prec_c C \vee E = C) \wedge E \preceq_c x\}$ $x =$ *epsilon* $\{E \in N.\ E \preceq_c x\}$ x
by (*metis* (*mono-tags*, *lifting*) *clause-order.order.strict-trans1 mem-Collect-eq*)
qed

ultimately show $(x, y) \in$ *epsilon* $\{D \in N.\ (\prec_c)^{==}\ D\ C\}$ C
unfolding *epsilon.simps*[*of - C*] **by** *simp*
qed

end

lemma (**in** *ground-ordering*) *Uniq-striclty-maximal-lit-in-ground-cls*:
$\exists_{\leq 1} L.$ *is-strictly-maximal-lit L C*
using *literal-order.Uniq-is-greatest-in-mset* **.**

lemma (**in** *ground-superposition-calculus*) *epsilon-eq-empty-or-singleton*:
epsilon N C $= \{\} \vee (\exists s\ t.$ *epsilon N C* $= \{(s, t)\})$
proof –
have $\exists_{\leq 1} (x, y).\ \exists C'.$
$C =$ *add-mset* (*Pos* (*Upair x y*)) $C' \wedge$ *is-strictly-maximal-lit* (*Pos* (*Upair x y*)) $C \wedge y \prec_t x$
by (*rule Uniq-prodI*)
(*metis Uniq-D Upair-inject literal-order.Uniq-is-greatest-in-mset term-order.min.absorb3*

term-order.min.absorb4 literal.inject(*1*))
hence *Uniq-epsilon*: $\exists_{\leq 1}$ (*x*, *y*). $\exists$ *C'*.
$C \in N \wedge$
C = *add-mset* (*Pos* (*Upair x y*)) *C'* $\wedge$ *select C* = {#} $\wedge$
is-strictly-maximal-lit (*Pos* (*Upair x y*)) *C* $\wedge$ $y \prec_t x \wedge$
(*let* $R_C = \bigcup D \in \{D \in N.\ D \prec_c C\}$. *epsilon* $\{E \in N.\ E \preceq_c D\}$ *D in*
$\neg$ *upair* ' (*rewrite-inside-gctxt* R_C)$^\downarrow$ $\Vdash$ *C* $\wedge$
$\neg$ *upair* ' (*rewrite-inside-gctxt* (*insert* (*x*, *y*) R_C))$^\downarrow$ $\Vdash$ *C'* $\wedge$
$x \in$ *NF* (*rewrite-inside-gctxt* R_C))
using *Uniq-antimono'*
by (*smt* (*verit*) *Uniq-def Uniq-prodI case-prod-conv*)
show *?thesis*
unfolding *epsilon.simps*[*of N C*]
using *Collect-eq-if-Uniq-prod*[*OF Uniq-epsilon*]
by (*smt* (*verit*, *best*) *Collect-cong Collect-empty-eq Uniq-def Uniq-epsilon case-prod-conv insertCI mem-Collect-eq*)
qed

lemma (**in** *ground-superposition-calculus*) *card-epsilon-le-one*:
card (*epsilon N C*) $\leq$ *1*
using *epsilon-eq-empty-or-singleton*[*of N C*]
by *auto*

definition (**in** *ground-superposition-calculus*) *rewrite-sys* **where**
rewrite-sys N C $\equiv$ ($\bigcup D \in \{D \in N.\ D \prec_c C\}$. *epsilon* $\{E \in N.\ E \preceq_c D\}$ *D*)

definition (**in** *ground-superposition-calculus*) *rewrite-sys'* **where**
rewrite-sys' N $\equiv$ ($\bigcup C \in N$. *epsilon N C*)

lemma (**in** *ground-superposition-calculus*) *rewrite-sys-alt*: *rewrite-sys'* $\{D \in N.\ D \prec_c C\}$ = *rewrite-sys N C*
unfolding *rewrite-sys'-def rewrite-sys-def*
proof (*rule SUP-cong*)
show $\{D \in N.\ D \prec_c C\} = \{D \in N.\ D \prec_c C\}$ **..**
next
show $\bigwedge x.\ x \in \{D \in N.\ D \prec_c C\} \Longrightarrow$ *epsilon* $\{D \in N.\ D \prec_c C\}$ *x* = *epsilon* $\{E \in N.\ (\prec_c)^{==}\ E\ x\}$ *x*
using *epsilon-filter-le-conv*
by (*smt* (*verit*, *best*) *Collect-cong clause-order.le-less-trans mem-Collect-eq*)
qed

lemma (**in** *ground-superposition-calculus*) *mem-epsilonE*:
assumes *rule-in*: *rule* $\in$ *epsilon N C*
obtains *l r C'* **where**
$C \in N$ **and**
rule = (*l*, *r*) **and**
C = *add-mset* (*Pos* (*Upair l r*)) *C'* **and**
select C = {#} **and**
is-strictly-maximal-lit (*Pos* (*Upair l r*)) *C* **and**

$r \prec_t l$ **and**
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$ $\Vdash$ C **and**
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^\downarrow$ $\Vdash$ C' **and**
$l \in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
using *rule-in*
unfolding *epsilon.simps*[*of N C*] *mem-Collect-eq Let-def rewrite-sys-def*
by (*metis* (*no-types*, *lifting*))

lemma (**in** *ground-superposition-calculus*) *mem-epsilon-iff*:
(*l*, *r*) $\in$ *epsilon N C* $\longleftrightarrow$
($\exists$ C'. $C \in N \wedge C =$ *add-mset* (*Pos* (*Upair l r*)) $C' \wedge$ *select* $C = \{\#\} \wedge$
is-strictly-maximal-lit (*Pos* (*Upair l r*)) $C \wedge r \prec_t l \wedge$
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys'* $\{D \in N.\ D \prec_c C\}$))$^\downarrow$ $\Vdash C \wedge$
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys'* $\{D \in N.\ D \prec_c C\}$)))$^\downarrow$
$\Vdash C' \wedge$
$l \in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys'* $\{D \in N.\ D \prec_c C\}$)))
(**is** *?LHS* $\longleftrightarrow$ *?RHS*)
proof (*rule iffI*)
assume *?LHS*
thus *?RHS*
using *rewrite-sys-alt*
by (*auto elim*: *mem-epsilonE*)
next
assume *?RHS*
thus *?LHS*
unfolding *epsilon.simps*[*of N C*] *mem-Collect-eq*
unfolding *rewrite-sys-alt rewrite-sys-def* **by** *auto*
qed

lemma (**in** *ground-superposition-calculus*) *rhs-lt-lhs-if-mem-rewrite-sys*:
assumes (*t1*, *t2*) $\in$ *rewrite-sys N C*
shows *t2* $\prec_t$ *t1*
using *assms*
unfolding *rewrite-sys-def*
by (*smt* (*verit*, *best*) *UN-iff mem-epsilonE prod.inject*)

lemma (**in** *ground-superposition-calculus*) *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys*:
assumes *rule-in*: (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
shows *t2* $\prec_t$ *t1*
proof −
from *rule-in* **obtain** *ctxt t1' t2'* **where**
(*t1*, *t2*) = (*ctxt*$\langle t1' \rangle_G$, *ctxt*$\langle t2' \rangle_G$) $\wedge$ (*t1'*, *t2'*) $\in$ *rewrite-sys N C*
unfolding *rewrite-inside-gctxt-def mem-Collect-eq*
by *auto*
thus *?thesis*
using *rhs-lt-lhs-if-mem-rewrite-sys*[*of t1' t2'*]
by (*metis Pair-inject less-trm-compatible-with-gctxt*)
qed

lemma (**in** *ground-superposition-calculus*) *rhs-lesseq-trm-lhs-if-mem-rtrancl-rewrite-inside-gctxt-rewrite-sys*:
assumes *rule-in*: $(t1, t2) \in (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^*$
shows $t2 \preceq_t t1$
using *rule-in*
proof (*induction t2 rule*: *rtrancl-induct*)
case *base*
show *?case*
by *order*
next
case (*step t2 t3*)
from *step.hyps* **have** $t3 \prec_t t2$
using *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys* **by** *metis*
with *step.IH* **show** *?case*
by *order*
qed

lemma *singleton-eq-CollectD*: $\{x\} = \{y.\ P\ y\} \Longrightarrow P\ x$
by *blast*

lemma *subset-Union-mem-CollectI*: $P\ x \Longrightarrow f\ x \subseteq (\bigcup y \in \{z.\ P\ z\}.\ f\ y)$
by *blast*

lemma (**in** *ground-superposition-calculus*) *rewrite-sys-subset-if-less-cls*:
$C \prec_c D \Longrightarrow$ *rewrite-sys N C* $\subseteq$ *rewrite-sys N D*
unfolding *rewrite-sys-def*
unfolding *epsilon-filter-le-conv*
by (*smt* (*verit*, *del-insts*) *SUP-mono clause-order.dual-order.strict-trans mem-Collect-eq subset-eq*)

lemma (**in** *ground-superposition-calculus*) *mem-rewrite-sys-if-less-cls*:
assumes $D \in N$ **and** $D \prec_c C$ **and** $(u, v) \in$ *epsilon N D*
shows $(u, v) \in$ *rewrite-sys N C*
unfolding *rewrite-sys-def UN-iff*
proof (*intro bexI*)
show $D \in \{D \in N.\ D \prec_c C\}$
using ‹$D \in N$› ‹$D \prec_c C$› **by** *simp*
next
show $(u, v) \in$ *epsilon* $\{E \in N.\ E \preceq_c D\}$ D
using ‹$(u, v) \in$ *epsilon N D*› *epsilon-filter-le-conv* **by** *simp*
qed

lemma (**in** *ground-superposition-calculus*) *less-trm-iff-less-cls-if-lhs-epsilon*:
assumes E_C: *epsilon N C* $= \{(s, t)\}$ **and** E_D: *epsilon N D* $= \{(u, v)\}$
shows $u \prec_t s \longleftrightarrow D \prec_c C$
proof −
from E_C **have** $(s, t) \in$ *epsilon N C*
by *simp*
then obtain C' **where**
$C \in N$ **and**

C-def: $C = $ *add-mset* (*Pos* (*Upair s t*)) C' **and**
is-strictly-maximal-lit (*Pos* (*Upair s t*)) C **and**
$t \prec_t s$ **and**
s-irreducible: $s \in$ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
by (*auto elim*!: *mem-epsilonE*)
hence $\forall L \in\# C'. L \prec_l$ *Pos* (*Upair s t*)
by (*simp add*: *literal-order.is-greatest-in-mset-iff*)

from E_D **obtain** D' **where**
$D \in N$ **and**
D-def: $D = $ *add-mset* (*Pos* (*Upair u v*)) D' **and**
is-strictly-maximal-lit (*Pos* (*Upair u v*)) D **and**
$v \prec_t u$
by (*auto simp*: *elim*: *epsilon.elims dest*: *singleton-eq-CollectD*)
hence $\forall L \in\# D'. L \prec_l$ *Pos* (*Upair u v*)
by (*simp add*: *literal-order.is-greatest-in-mset-iff*)

show *?thesis*
proof (*rule iffI*)
assume $u \prec_t s$
moreover hence $v \prec_t s$
using ‹$v \prec_t u$› **by** *order*
ultimately have *multp* ($\prec_t$) {#*u*, *v*#} {#*s*, *t*#}
using *one-step-implies-multp*[*of* {#*s*, *t*#} {#*u*, *v*#} - {#}] **by** *simp*
hence *Pos* (*Upair u v*) $\prec_l$ *Pos* (*Upair s t*)
by (*simp add*: *less-lit-def*)
moreover hence $\forall L \in\# D'. L \prec_l$ *Pos* (*Upair s t*)
using ‹$\forall L \in\# D'. L \prec_l$ *Pos* (*Upair u v*)›
by (*meson literal-order.transp-on-less transpD*)
ultimately show $D \prec_c C$
using *one-step-implies-multp*[*of C D* - {#}] *less-cls-def*
by (*simp add*: *D-def C-def*)
next
assume $D \prec_c C$
have $(u, v) \in$ *rewrite-sys N C*
using E_D ‹$D \in N$› ‹$D \prec_c C$› *mem-rewrite-sys-if-less-cls* **by** *auto*
hence $(u, v) \in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
by *blast*
hence $s \neq u$
using *s-irreducible*
by *auto*
moreover have $\neg (s \prec_t u)$
proof (*rule notI*)
assume $s \prec_t u$
moreover hence $t \prec_t u$
using ‹$t \prec_t s$› **by** *order*
ultimately have *multp* ($\prec_t$) {#*s*, *t*#} {#*u*, *v*#}
using *one-step-implies-multp*[*of* {#*u*, *v*#} {#*s*, *t*#} - {#}] **by** *simp*
hence *Pos* (*Upair s t*) $\prec_l$ *Pos* (*Upair u v*)

```
        by (simp add: less-lit-def)
      moreover hence ∀ L ∈# C'. L ≺_l Pos (Upair u v)
        using ‹∀ L ∈# C'. L ≺_l Pos (Upair s t)›
        by (meson literal-order.transp-on-less transpD)
      ultimately have C ≺_c D
        using one-step-implies-multp[of D C - {#}] less-cls-def
        by (simp add: D-def C-def)
      thus False
        using ‹D ≺_c C› by order
    qed
    ultimately show u ≺_t s
      by order
  qed
qed

lemma (in ground-superposition-calculus) termination-rewrite-sys: wf ((rewrite-sys
N C)^-1)
proof (rule wf-if-convertible-to-wf)
  show wf {(x, y). x ≺_t y}
    using wfP-less-trm
    by (simp add: wfP-def)
next
  fix t s
  assume (t, s) ∈ (rewrite-sys N C)^-1
  hence (s, t) ∈ rewrite-sys N C
    by simp
  then obtain D where D ≺_c C and (s, t) ∈ epsilon N D
    unfolding rewrite-sys-def using epsilon-filter-le-conv by blast
  hence t ≺_t s
    by (auto elim: mem-epsilonE)
  thus (t, s) ∈ {(x, y). x ≺_t y}
    by (simp add: )
qed

lemma (in ground-superposition-calculus) termination-Union-rewrite-sys:
  wf ((⋃ D ∈ N. rewrite-sys N D)^-1)
proof (rule wf-if-convertible-to-wf)
  show wf {(x, y). x ≺_t y}
    using wfP-less-trm
    by (simp add: wfP-def)
next
  fix t s
  assume (t, s) ∈ (⋃ D ∈ N. rewrite-sys N D)^-1
  hence (s, t) ∈ (⋃ D ∈ N. rewrite-sys N D)
    by simp
  then obtain C where C ∈ N (s, t) ∈ rewrite-sys N C
    by auto
  then obtain D where D ≺_c C and (s, t) ∈ epsilon N D
    unfolding rewrite-sys-def using epsilon-filter-le-conv by blast
```

hence $t \prec_t s$
by (*auto elim*: *mem-epsilonE*)
thus $(t, s) \in \{(x, y).\ x \prec_t y\}$
by *simp*
qed

lemma (**in** *ground-superposition-calculus*) *no-crit-pairs*:
$\{(t1, t2) \in \textit{ground-critical-pairs}\ (\bigcup\ (\textit{epsilon N2 ` N})).\ t1 \neq t2\} = \{\}$
proof (*rule ccontr*)
assume $\{(t1, t2).$
$(t1, t2) \in \textit{ground-critical-pairs}\ (\bigcup\ (\textit{epsilon N2 ` N})) \wedge t1 \neq t2\} \neq \{\}$
then obtain *ctxt l r1 r2* **where**
$(\textit{ctxt}\langle r2\rangle_G, r1) \in \textit{ground-critical-pairs}\ (\bigcup\ (\textit{epsilon N2 ` N}))$ **and**
$\textit{ctxt}\langle r2\rangle_G \neq r1$ **and**
rule1-in: $(\textit{ctxt}\langle l\rangle_G, r1) \in \bigcup\ (\textit{epsilon N2 ` N})$ **and**
rule2-in: $(l, r2) \in \bigcup\ (\textit{epsilon N2 ` N})$
unfolding *ground-critical-pairs-def mem-Collect-eq* **by** *blast*

from *rule1-in rule2-in* **obtain** *C1 C2* **where**
$C1 \in N$ **and** *rule1-in'*: $(\textit{ctxt}\langle l\rangle_G, r1) \in \textit{epsilon N2 C1}$ **and**
$C2 \in N$ **and** *rule2-in'*: $(l, r2) \in \textit{epsilon N2 C2}$
by *auto*

from *rule1-in'* **obtain** *C1'* **where**
C1-def: $C1 = \textit{add-mset}\ (\textit{Pos}\ (\textit{Upair}\ \textit{ctxt}\langle l\rangle_G\ r1))\ C1'$ **and**
C1-max: *is-strictly-maximal-lit* $(\textit{Pos}\ (\textit{Upair}\ \textit{ctxt}\langle l\rangle_G\ r1))\ C1$ **and**
$r1 \prec_t \textit{ctxt}\langle l\rangle_G$ **and**
l1-irreducible: $\textit{ctxt}\langle l\rangle_G \in \textit{NF}\ (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys N2 C1}))$
by (*auto elim*: *mem-epsilonE*)

from *rule2-in'* **obtain** *C2'* **where**
C2-def: $C2 = \textit{add-mset}\ (\textit{Pos}\ (\textit{Upair}\ l\ r2))\ C2'$ **and**
C2-max: *is-strictly-maximal-lit* $(\textit{Pos}\ (\textit{Upair}\ l\ r2))\ C2$ **and**
$r2 \prec_t l$
by (*auto elim*: *mem-epsilonE*)

have $\textit{epsilon N2 C1} = \{(\textit{ctxt}\langle l\rangle_G, r1)\}$
using *rule1-in' epsilon-eq-empty-or-singleton* **by** *fastforce*

have $\textit{epsilon N2 C2} = \{(l, r2)\}$
using *rule2-in' epsilon-eq-empty-or-singleton* **by** *fastforce*

show *False*
proof ($\textit{cases ctxt} = \Box_G$)
case *True*
hence $\neg\ (\textit{ctxt}\langle l\rangle_G \prec_t l)$ **and** $\neg\ (l \prec_t \textit{ctxt}\langle l\rangle_G)$
by (*simp-all add*: *irreflpD*)
hence $\neg\ (C1 \prec_c C2)$ **and** $\neg\ (C2 \prec_c C1)$
using ‹$\textit{epsilon N2 C1} = \{(\textit{ctxt}\langle l\rangle_G, r1)\}$› ‹$\textit{epsilon N2 C2} = \{(l, r2)\}$›

less-trm-iff-less-cls-if-lhs-epsilon
by *simp-all*
hence *C1* = *C2*
by *order*
hence *r1* = *r2*
using ‹*epsilon N2 C1* = {(*ctxt*⟨*l*⟩$_G$, *r1*)}› ‹*epsilon N2 C2* = {(*l*, *r2*)}› **by** *simp*
moreover have *r1* ≠ *r2*
using ‹*ctxt*⟨*r2*⟩$_G$ ≠ *r1*›
unfolding ‹*ctxt* = □$_G$›
by *simp*
ultimately show *?thesis*
by *contradiction*
next
case *False*
hence *l* $\prec_t$ *ctxt*⟨*l*⟩$_G$
by (*metis less-trm-if-subterm*)
hence *C2* $\prec_c$ *C1*
using ‹*epsilon N2 C1* = {(*ctxt*⟨*l*⟩$_G$, *r1*)}› ‹*epsilon N2 C2* = {(*l*, *r2*)}›
less-trm-iff-less-cls-if-lhs-epsilon
by *simp*
have (*l*, *r2*) ∈ *rewrite-sys N2 C1*
by (*metis* ‹*C2* $\prec_c$ *C1*› ‹*epsilon N2 C2* = {(*l*, *r2*)}› *mem-epsilonE mem-rewrite-sys-if-less-cls singletonI*)
hence (*ctxt*⟨*l*⟩$_G$, *ctxt*⟨*r2*⟩$_G$) ∈ *rewrite-inside-gctxt* (*rewrite-sys N2 C1*)
by *auto*
thus *False*
using *l1-irreducible* **by** *auto*
qed
qed

lemma (**in** *ground-superposition-calculus*) *WCR-Union-rewrite-sys*:
WCR (*rewrite-inside-gctxt* (⋃*D* ∈ *N*. *epsilon N2 D*))
unfolding *ground-critical-pair-theorem*
proof (*intro subsetI ballI*)
fix *tuple*
assume *tuple-in*: *tuple* ∈ *ground-critical-pairs* (⋃ (*epsilon N2* ‘ *N*))
then obtain *t1 t2* **where** *tuple-def*: *tuple* = (*t1*, *t2*)
by *fastforce*

moreover have (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (⋃ (*epsilon N2* ‘ *N*)))$^{\downarrow}$ **if** *t1* = *t2*
using *that* **by** *auto*

moreover have *False* **if** *t1* ≠ *t2*
using *that tuple-def tuple-in no-crit-pairs* **by** *simp*

ultimately show *tuple* ∈ (*rewrite-inside-gctxt* (⋃ (*epsilon N2* ‘ *N*)))$^{\downarrow}$
by (*cases t1* = *t2*) *simp-all*

qed

lemma (**in** *ground-superposition-calculus*)
  assumes
    $D \preceq_c C$ **and**
    E_C*-eq*: *epsilon N C* = {(*s*, *t*)} **and**
    L-in: $L \in\# D$ **and**
    topmost-trms-of-L: *mset-uprod* (*atm-of L*) = {#*u*, *v*#}
  shows
    lesseq-trm-if-pos: *is-pos L* $\Longrightarrow u \preceq_t s$ **and**
    less-trm-if-neg: *is-neg L* $\Longrightarrow u \prec_t s$
proof −
  from E_C*-eq* **have** $(s, t) \in$ *epsilon N C*
    by *simp*
  then obtain C' **where**
    C-def: *C* = *add-mset* (*Pos* (*Upair s t*)) C' **and**
    C-max-lit: *is-strictly-maximal-lit* (*Pos* (*Upair s t*)) *C* **and**
    $t \prec_t s$
    by (*auto elim*: *mem-epsilonE*)

  have *Pos* (*Upair s t*) $\prec_l$ *L* **if** *is-pos L* **and** $\neg\ u \preceq_t s$
  proof −
    from *that*(*2*) **have** $s \prec_t u$
      by *order*
    hence *multp* $(\prec_t)$ {#*s*, *t*#} {#*u*, *v*#}
      using ‹$t \prec_t s$›
     by (*smt* (*verit*, *del-insts*) *add.right-neutral empty-iff insert-iff one-step-implies-multp*
       set-mset-add-mset-insert set-mset-empty transpD transp-less-trm union-mset-add-mset-right)
    with *that*(*1*) **show** *Pos* (*Upair s t*) $\prec_l$ *L*
      using *topmost-trms-of-L*
      by (*cases L*) (*simp-all add*: *less-lit-def*)
  qed

  moreover have *Pos* (*Upair s t*) $\prec_l$ *L* **if** *is-neg L* **and** $\neg\ u \prec_t s$
  proof −
    from *that*(*2*) **have** $s \preceq_t u$
      by *order*
    hence *multp* $(\prec_t)$ {#*s*, *t*#} {#*u*, *v*, *u*, *v*#}
      using ‹$t \prec_t s$›
     by (*smt* (*z3*) *add-mset-add-single add-mset-remove-trivial add-mset-remove-trivial-iff*
       empty-not-add-mset insert-DiffM insert-noteq-member one-step-implies-multp
reflclp-iff
       transp-def transp-less-trm union-mset-add-mset-left union-mset-add-mset-right)
    with *that*(*1*) **show** *Pos* (*Upair s t*) $\prec_l$ *L*
      using *topmost-trms-of-L*
      by (*cases L*) (*simp-all add*: *less-lit-def*)
  qed

  moreover have *False* **if** *Pos* (*Upair s t*) $\prec_l$ *L*

```
  proof –
    have C ≺c D
      unfolding less-cls-def
    proof (rule multp-if-maximal-of-lhs-is-less)
      show Pos (Upair s t) ∈# C
        by (simp add: C-def)
    next
      show L ∈# D
        using L-in by simp
    next
      show is-maximal-lit (Pos (Upair s t)) C
        using C-max-lit by auto
    next
      show Pos (Upair s t) ≺l L
        using that .
    qed simp-all
    with ‹D ≼c C› show False
      by order
  qed

  ultimately show is-pos L ⟹ u ≼t s and is-neg L ⟹ u ≺t s
    by argo+
qed

lemma (in ground-ordering) less-trm-const-lhs-if-mem-rewrite-inside-gctxt:
  fixes t t1 t2 r
  assumes
    rule-in: (t1, t2) ∈ rewrite-inside-gctxt r and
    ball-lt-lhs: ⋀t1 t2. (t1, t2) ∈ r ⟹ t ≺t t1
  shows t ≺t t1
proof –
  from rule-in obtain ctxt t1' t2' where
    rule-in': (t1', t2') ∈ r and
    l-def: t1 = ctxt⟨t1'⟩G and
    r-def: t2 = ctxt⟨t2'⟩G
    unfolding rewrite-inside-gctxt-def by fast

  show ?thesis
    using ball-lt-lhs[OF rule-in'] lesseq-trm-if-subtermeq[of t1' ctxt] l-def by order
qed

lemma (in ground-superposition-calculus) split-Union-epsilon:
  assumes D-in: D ∈ N
  shows (⋃C ∈ N. epsilon N C) =
    rewrite-sys N D ∪ epsilon N D ∪ (⋃C ∈ {C ∈ N. D ≺c C}. epsilon N C)
proof –
  have N = {C ∈ N. C ≺c D} ∪ {D} ∪ {C ∈ N. D ≺c C}
  proof (rule partition-set-around-element)
    show totalp-on N (≺c)
```

using *clause-order.totalp-on-less* .
next
show $D \in N$
using *D-in* **by** *simp*
qed
hence $(\bigcup C \in N.\ epsilon\ N\ C) =$
$(\bigcup C \in \{C \in N.\ C \prec_c D\}.\ epsilon\ N\ C) \cup epsilon\ N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ epsilon\ N\ C)$
by *auto*
thus $(\bigcup C \in N.\ epsilon\ N\ C) =$
$rewrite\text{-}sys\ N\ D \cup epsilon\ N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.\ epsilon\ N\ C)$
using *epsilon-filter-le-conv rewrite-sys-def* **by** *simp*
qed

lemma (**in** *ground-superposition-calculus*) *split-Union-epsilon'*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.\ epsilon\ N\ C) = rewrite\text{-}sys\ N\ D \cup (\bigcup C \in \{C \in N.\ D \preceq_c C\}.\ epsilon\ N\ C)$
using *split-Union-epsilon*[*OF D-in*] *D-in* **by** *auto*

lemma (**in** *ground-superposition-calculus*) *split-rewrite-sys*:
assumes $C \in N$ **and** *D-in*: $D \in N$ **and** $D \prec_c C$
shows $rewrite\text{-}sys\ N\ C = rewrite\text{-}sys\ N\ D \cup (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}.\ epsilon\ N\ C')$
proof −
have $\{D \in N.\ D \prec_c C\} =$
$\{y \in \{D \in N.\ D \prec_c C\}.\ y \prec_c D\} \cup \{D\} \cup \{y \in \{D \in N.\ D \prec_c C\}.\ D \prec_c y\}$
proof (*rule partition-set-around-element*)
show *totalp-on* $\{D \in N.\ D \prec_c C\}\ (\prec_c)$
using *clause-order.totalp-on-less* .
next
from *D-in* ‹$D \prec_c C$› **show** $D \in \{D \in N.\ D \prec_c C\}$
by *simp*
qed
also have $\ldots = \{x \in N.\ x \prec_c C \wedge x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
by *auto*
also have $\ldots = \{x \in N.\ x \prec_c D\} \cup \{D\} \cup \{x \in N.\ D \prec_c x \wedge x \prec_c C\}$
using ‹$D \prec_c C$› *clause-order.transp-on-less*
by (*metis* (*no-types*, *opaque-lifting*) *transpD*)
finally have *Collect-N-lt-C*: $\{x \in N.\ x \prec_c C\} = \{x \in N.\ x \prec_c D\} \cup \{x \in N.\ D \preceq_c x \wedge x \prec_c C\}$
by *auto*

have $rewrite\text{-}sys\ N\ C = (\bigcup C' \in \{D \in N.\ D \prec_c C\}.\ epsilon\ N\ C')$
using *epsilon-filter-le-conv*
by (*simp add*: *rewrite-sys-def*)
also have $\ldots = (\bigcup C' \in \{x \in N.\ x \prec_c D\}.\ epsilon\ N\ C') \cup (\bigcup C' \in \{x \in N.\ D$

$\preceq_c$ $x \wedge x \prec_c C\}$. *epsilon N C′*)
    unfolding *Collect-N-lt-C* **by** *simp*
  finally show *rewrite-sys N C* = *rewrite-sys N D* ∪ ⋃ (*epsilon N* ‘ {*C′* ∈ *N*. *D* $\preceq_c$ *C′* ∧ *C′* $\prec_c$ *C*})
    using *epsilon-filter-le-conv*
    unfolding *rewrite-sys-def* **by** *simp*
qed

lemma (**in** *ground-ordering*) *mem-join-union-iff-mem-join-lhs′*:
  assumes
    ball-R_1-rhs-lt-lhs: ⋀*t1 t2*. (*t1*, *t2*) ∈ R_1 ⟹ *t2* $\prec_t$ *t1* **and**
    ball-R_2-lt-lhs: ⋀*t1 t2*. (*t1*, *t2*) ∈ R_2 ⟹ *s* $\prec_t$ *t1* ∧ *t* $\prec_t$ *t1*
  shows (*s*, *t*) ∈ $(R_1 \cup R_2)^{\downarrow}$ ⟷ (*s*, *t*) ∈ $R_1^{\downarrow}$
proof −
  have *ball-R_1-rhs-lt-lhs′*: (*t1*, *t2*) ∈ R_1^* ⟹ *t2* $\preceq_t$ *t1* **for** *t1 t2*
  proof (*induction t2 rule*: *rtrancl-induct*)
    case *base*
    show *?case*
      by *order*
  next
    case (*step y z*)
    thus *?case*
      using *ball-R_1-rhs-lt-lhs*
      by (*metis reflclp-iff transpD transp-less-trm*)
  qed

  show *?thesis*
  proof (*rule mem-join-union-iff-mem-join-lhs*)
    fix *u* **assume** (*s*, *u*) ∈ R_1^*
    hence *u* $\preceq_t$ *s*
      using *ball-R_1-rhs-lt-lhs′* **by** *metis*

    show *u* ∉ *Domain* R_2
    proof (*rule notI*)
      assume *u* ∈ *Domain* R_2
      then obtain *u′* **where** (*u*, *u′*) ∈ R_2
        by *auto*
      hence *s* $\prec_t$ *u*
        using *ball-R_2-lt-lhs* **by** *simp*
      with ‹*u* $\preceq_t$ *s*› **show** *False*
        by *order*
    qed
  next
    fix *u* **assume** (*t*, *u*) ∈ R_1^*
    hence *u* $\preceq_t$ *t*
      using *ball-R_1-rhs-lt-lhs′* **by** *simp*

    show *u* ∉ *Domain* R_2
    proof (*rule notI*)

assume $u \in \mathit{Domain}\ R_2$
then obtain u' **where** $(u,\ u') \in R_2$
by *auto*
hence $t \prec_t u$
using *ball-*R_2*-lt-lhs* **by** *simp*
with ‹$u \preceq_t t$› **show** *False*
by *order*
qed
qed
qed

lemma (**in** *ground-ordering*) *mem-join-union-iff-mem-join-rhs*$'$:
assumes
ball-R_1*-rhs-lt-lhs*: $\bigwedge t1\ t2.\ (t1,\ t2) \in R_2 \Longrightarrow t2 \prec_t t1$ **and**
ball-R_2*-lt-lhs*: $\bigwedge t1\ t2.\ (t1,\ t2) \in R_1 \Longrightarrow s \prec_t t1 \wedge t \prec_t t1$
shows $(s,\ t) \in (R_1 \cup R_2)^{\downarrow} \longleftrightarrow (s,\ t) \in {R_2}^{\downarrow}$
using *assms mem-join-union-iff-mem-join-lhs*$'$
by (*metis* (*no-types*, *opaque-lifting*) *sup-commute*)

lemma (**in** *ground-ordering*) *mem-join-union-iff-mem-join-lhs*$''$:
assumes
Range-R_1*-lt-Domain-*R_2: $\bigwedge t1\ t2.\ t1 \in \mathit{Range}\ R_1 \Longrightarrow t2 \in \mathit{Domain}\ R_2 \Longrightarrow t1$
$\prec_t t2$ **and**
s-lt-Domain-R_2: $\bigwedge t2.\ t2 \in \mathit{Domain}\ R_2 \Longrightarrow s \prec_t t2$ **and**
t-lt-Domain-R_2: $\bigwedge t2.\ t2 \in \mathit{Domain}\ R_2 \Longrightarrow t \prec_t t2$
shows $(s,\ t) \in (R_1 \cup R_2)^{\downarrow} \longleftrightarrow (s,\ t) \in {R_1}^{\downarrow}$
proof (*rule mem-join-union-iff-mem-join-lhs*)
fix u **assume** $(s,\ u) \in {R_1}^*$
hence $u = s \vee u \in \mathit{Range}\ R_1$
by (*meson Range.intros rtrancl.cases*)
thus $u \notin \mathit{Domain}\ R_2$
using *Range-*R_1*-lt-Domain-*R_2 *s-lt-Domain-*R_2
by (*metis irreflpD term-order.irreflp-on-less*)
next
fix u **assume** $(t,\ u) \in {R_1}^*$
hence $u = t \vee u \in \mathit{Range}\ R_1$
by (*meson Range.intros rtrancl.cases*)
thus $u \notin \mathit{Domain}\ R_2$
using *Range-*R_1*-lt-Domain-*R_2 *t-lt-Domain-*R_2
by (*metis irreflpD term-order.irreflp-on-less*)
qed

lemma (**in** *ground-superposition-calculus*) *lift-entailment-to-Union*:
fixes $N\ D$
defines $R_D \equiv$ *rewrite-sys* $N\ D$
assumes
D-in: $D \in N$ **and**
R_D*-entails-D*: *upair* ‘ (*rewrite-inside-gctxt* $R_D)^{\downarrow} \Vdash D$
shows

```
    upair ‘ (rewrite-inside-gctxt (⋃D ∈ N. epsilon N D))↓ ⊫ D and
    ⋀C. C ∈ N ⟹ D ≺c C ⟹ upair ‘ (rewrite-inside-gctxt (rewrite-sys N C))↓
⊫ D
proof −
  from RD-entails-D obtain L s t where
    L-in: L ∈# D and
    L-eq-disj-L-eq: L = Pos (Upair s t) ∧ (s, t) ∈ (rewrite-inside-gctxt RD)↓ ∨
     L = Neg (Upair s t) ∧ (s, t) ∉ (rewrite-inside-gctxt RD)↓
    unfolding true-cls-def true-lit-iff
   by (metis (no-types, opaque-lifting) image-iff prod.case surj-pair uprod-exhaust)

  from L-eq-disj-L-eq show
    upair ‘ (rewrite-inside-gctxt (⋃D ∈ N. epsilon N D))↓ ⊫ D and
    ⋀C. C ∈ N ⟹ D ≺c C ⟹ upair ‘ (rewrite-inside-gctxt (rewrite-sys N C))↓
⊫ D
    unfolding atomize-all atomize-conj atomize-imp
  proof (elim disjE conjE)
    assume L-def: L = Pos (Upair s t) and (s, t) ∈ (rewrite-inside-gctxt RD)↓
    have RD ⊆ (⋃D ∈ N. epsilon N D) and
      ∀C. C ∈ N ⟶ D ≺c C ⟶ RD ⊆ rewrite-sys N C
      unfolding RD-def rewrite-sys-def
      using D-in clause-order.transp-on-less[THEN transpD]
      using epsilon-filter-le-conv
      by (auto intro: Collect-mono)
    hence rewrite-inside-gctxt RD ⊆ rewrite-inside-gctxt (⋃D ∈ N. epsilon N D)
and
      ∀C. C ∈ N ⟶ D ≺c C ⟶ rewrite-inside-gctxt RD ⊆ rewrite-inside-gctxt
(rewrite-sys N C)
      by (auto intro!: rewrite-inside-gctxt-mono)
    hence (s, t) ∈ (rewrite-inside-gctxt (⋃D ∈ N. epsilon N D))↓ and
     ∀C. C ∈ N ⟶ D ≺c C ⟶ (s, t) ∈ (rewrite-inside-gctxt (rewrite-sys N C))↓
      by (auto intro!: join-mono intro: set-mp[OF - ‹(s, t) ∈ (rewrite-inside-gctxt
RD)↓›])
    thus upair ‘ (rewrite-inside-gctxt (⋃ (epsilon N ‘ N)))↓ ⊫ D ∧
       (∀C. C ∈ N ⟶ D ≺c C ⟶ upair ‘ (rewrite-inside-gctxt (rewrite-sys N
C))↓ ⊫ D)
      unfolding true-cls-def true-lit-iff
      using L-in L-def by blast
  next
    have (t1, t2) ∈ RD ⟹ t2 ≺t t1 for t1 t2
      by (auto simp: RD-def rewrite-sys-def elim: mem-epsilonE)
    hence ball-RD-rhs-lt-lhs: (t1, t2) ∈ rewrite-inside-gctxt RD ⟹ t2 ≺t t1 for
t1 t2
    by (smt (verit, ccfv-SIG) Pair-inject less-trm-compatible-with-gctxt mem-Collect-eq
        rewrite-inside-gctxt-def)

    assume L-def: L = Neg (Upair s t) and (s, t) ∉ (rewrite-inside-gctxt RD)↓

    have (s, t) ∈ (rewrite-inside-gctxt RD ∪ rewrite-inside-gctxt (⋃C ∈ {C ∈ N.
```

$D \preceq_c C\}.$ *epsilon N C*$))^{\downarrow} \longleftrightarrow$
$(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D)^{\downarrow}$
proof (*rule mem-join-union-iff-mem-join-lhs′*)
show $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* $R_D \Longrightarrow$ *t2* $\prec_t$ *t1*
using *ball-*R_D*-rhs-lt-lhs* **by** *simp*
next
have *ball-Rinf-minus-lt-lhs*: $s \prec_t$ *fst rule* $\wedge$ $t \prec_t$ *fst rule*
if *rule-in*: *rule* $\in (\bigcup C \in \{C \in N.\ D \preceq_c C\}.$ *epsilon N C*)
for *rule*
proof −
from *rule-in* **obtain** C **where**
$C \in N$ **and** $D \preceq_c C$ **and** *rule* $\in$ *epsilon N C*
by *auto*

have *epsilon-C-eq*: *epsilon N C* = {(*fst rule*, *snd rule*)}
using ‹*rule* $\in$ *epsilon N C*› *epsilon-eq-empty-or-singleton* **by** *force*

show *?thesis*
using *less-trm-if-neg*[*OF* ‹$D \preceq_c C$› *epsilon-C-eq L-in*]
by (*simp add*: *L-def*)
qed

show $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ $\{C \in N.\ (\prec_c)^{==}$
$D\ C\}$)) $\Longrightarrow$
$s \prec_t$ *t1* $\wedge$ $t \prec_t$ *t1*
using *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*
using *ball-Rinf-minus-lt-lhs*
by *force*
qed

moreover have
$(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D \cup$ *rewrite-inside-gctxt* $(\bigcup C' \in \{C' \in N.\ D$
$\preceq_c C' \wedge C' \prec_c C\}.$ *epsilon N C′*$))^{\downarrow} \longleftrightarrow$
$(s,\ t) \in$ (*rewrite-inside-gctxt* $R_D)^{\downarrow}$
if $C \in N$ **and** $D \prec_c C$
for C
proof (*rule mem-join-union-iff-mem-join-lhs′*)
show $\bigwedge$*t1 t2*. (*t1*, *t2*) $\in$ *rewrite-inside-gctxt* $R_D \Longrightarrow$ *t2* $\prec_t$ *t1*
using *ball-*R_D*-rhs-lt-lhs* **by** *simp*
next
have *ball-lt-lhs*: $s \prec_t$ *t1* $\wedge$ $t \prec_t$ *t1*
if $C \in N$ **and** $D \prec_c C$ **and**
rule-in: (*t1*, *t2*) $\in (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C' \prec_c C\}.$ *epsilon N C′*)
for *C t1 t2*
proof −
from *rule-in* **obtain** C' **where**
$C' \in N$ **and** $D \preceq_c C'$ **and** $C' \prec_c C$ **and** (*t1*, *t2*) $\in$ *epsilon N C′*
by (*auto simp*: *rewrite-sys-def*)

have *epsilon-C′-eq*: *epsilon N C′* = {(*t1*, *t2*)}
using ‹(*t1*, *t2*) ∈ *epsilon N C′*› *epsilon-eq-empty-or-singleton* **by** *force*

show *?thesis*
using *less-trm-if-neg*[*OF* ‹$D \preceq_c C'$› *epsilon-C′-eq L-in*]
by (*simp add*: *L-def*)
qed

show $\bigwedge$*t1 t2*. (*t1*, *t2*) ∈ *rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ {*C′* ∈ *N*. $(\prec_c)^{==}$ *D C′* ∧ $C' \prec_c C$})) ⟹
$s \prec_t t1 \wedge t \prec_t t1$
using *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*
using *ball-lt-lhs*[*OF that*(*1*,*2*)]
by (*metis* (*no-types*, *lifting*))
qed

ultimately have (*s*, *t*) ∉ (*rewrite-inside-gctxt* R_D ∪ *rewrite-inside-gctxt* ($\bigcup$ *C* ∈ {*C* ∈ *N*. $D \preceq_c C$}. *epsilon N C*))$^\downarrow$ **and**
∀ *C*. *C* ∈ *N* ⟶ $D \prec_c C$ ⟶
(*s*, *t*) ∉ (*rewrite-inside-gctxt* R_D ∪ *rewrite-inside-gctxt* ($\bigcup$ *C′* ∈ {*C′* ∈ *N*. $D \preceq_c C' \wedge C' \prec_c C$}. *epsilon N C′*))$^\downarrow$
using ‹(*s*, *t*) ∉ (*rewrite-inside-gctxt* R_D)$^\downarrow$› **by** *simp-all*
hence (*s*, *t*) ∉ (*rewrite-inside-gctxt* ($\bigcup$ *D* ∈ *N*. *epsilon N D*))$^\downarrow$ **and**
∀ *C*. *C* ∈ *N* ⟶ $D \prec_c C$ ⟶ (*s*, *t*) ∉ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
using *split-Union-epsilon′*[*OF D-in*, *folded* R_D*-def*]
using *split-rewrite-sys*[*OF - D-in*, *folded* R_D*-def*]
by (*simp-all add*: *rewrite-inside-gctxt-union*)
hence (*Upair s t*) ∉ *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup$ *D* ∈ *N*. *epsilon N D*))$^\downarrow$ **and**
∀ *C*. *C* ∈ *N* ⟶ $D \prec_c C$ ⟶ (*Upair s t*) ∉ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
unfolding *atomize-conj*
by (*meson sym-join true-lit-simps*(*2*) *true-lit-uprod-iff-true-lit-prod*(*2*))
thus *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ *N*)))$^\downarrow$ ⊫ *D* ∧
(∀ *C*. *C* ∈ *N* ⟶ $D \prec_c C$ ⟶ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$ ⊫ *D*)
unfolding *true-cls-def true-lit-iff*
using *L-in L-def* **by** *metis*
qed
qed

lemma (**in** *ground-superposition-calculus*)
assumes *productive*: *epsilon N C* = {(*l*, *r*)}
shows
true-cls-if-productive-epsilon:
upair ‘ (*rewrite-inside-gctxt* ($\bigcup$ *D* ∈ *N*. *epsilon N D*))$^\downarrow$ ⊫ *C*
$\bigwedge$*D*. *D* ∈ *N* ⟹ $C \prec_c D$ ⟹ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ ⊫ *C* **and**
false-cls-if-productive-epsilon:

$\neg$ *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*))$^{\downarrow}$ $\models$ *C* − {#*Pos* (*Upair l r*)#}
$\bigwedge D$. $D \in N \Longrightarrow C \prec_c D \Longrightarrow \neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\models$ *C* − {#*Pos* (*Upair l r*)#}
proof −
from *productive* **have** (*l*, *r*) ∈ *epsilon N C*
by *simp*
then obtain *C′* **where**
C-in: *C* ∈ *N* **and**
C-def: *C* = *add-mset* (*Pos* (*Upair l r*)) *C′* **and**
select C = {#} **and**
is-strictly-maximal-lit (*Pos* (*Upair l r*)) *C* **and**
$r \prec_t l$ **and**
e: ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\models$ *C* **and**
f: ¬ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$ $\models$ *C′* **and**
l ∈ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
by (*rule mem-epsilonE*) *blast*

have (*l*, *r*) ∈ (*rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*))$^{\downarrow}$
using *C-in* ‹(*l*, *r*) ∈ *epsilon N C*› *mem-rewrite-inside-gctxt-if-mem-rewrite-rules* **by** *blast*
thus *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*))$^{\downarrow}$ $\models$ *C*
using *C-def* **by** *blast*

have *rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*) =
rewrite-inside-gctxt (*rewrite-sys N C* ∪ *epsilon N C* ∪ ($\bigcup D \in \{D \in N.\ C \prec_c D\}$. *epsilon N D*))
using *split-Union-epsilon*[*OF C-in*] **by** *simp*
also have ... =
rewrite-inside-gctxt (*rewrite-sys N C* ∪ *epsilon N C*) ∪
rewrite-inside-gctxt ($\bigcup D \in \{D \in N.\ C \prec_c D\}$. *epsilon N D*)
by (*simp add*: *rewrite-inside-gctxt-union*)
finally have *rewrite-inside-gctxt-Union-epsilon-eq*:
rewrite-inside-gctxt ($\bigcup D \in N$. *epsilon N D*) =
rewrite-inside-gctxt (*insert* (*l*, *r*) (*rewrite-sys N C*)) ∪
rewrite-inside-gctxt ($\bigcup D \in \{D \in N.\ C \prec_c D\}$. *epsilon N D*)
unfolding *productive* **by** *simp*

have *mem-join-union-iff-mem-lhs*:(*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)) ∪
rewrite-inside-gctxt ($\bigcup D \in \{D \in N.\ C \prec_c D\}$. *epsilon N D*))$^{\downarrow}$ ⟷
(*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$
if $t1 \preceq_t l$ **and** $t2 \preceq_t l$
for *t1 t2*
proof (*rule mem-join-union-iff-mem-join-lhs′*)
fix *s1 s2* **assume** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*))

moreover have $s2 \prec_t s1$ **if** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* {(*l*, *r*)}

proof (*rule rhs-lt-lhs-if-rule-in-rewrite-inside-gctxt*[*OF that*])
show $\bigwedge$*s1 s2*. (*s1*, *s2*) ∈ {(*l*, *r*)} ⟹ *s2* $\prec_t$ *s1*
using ‹*r* $\prec_t$ *l*› **by** *simp*
qed *simp-all*

moreover have *s2* $\prec_t$ *s1* **if** (*s1*, *s2*) ∈ *rewrite-inside-gctxt* (*rewrite-sys N C*)
proof (*rule rhs-lt-lhs-if-rule-in-rewrite-inside-gctxt*[*OF that*])
show $\bigwedge$*s1 s2*. (*s1*, *s2*) ∈ *rewrite-sys N C* ⟹ *s2* $\prec_t$ *s1*
by (*simp add*: *rhs-lt-lhs-if-mem-rewrite-sys*)
qed *simp*

ultimately show *s2* $\prec_t$ *s1*
using *rewrite-inside-gctxt-union*[*of* {(*l*, *r*)}, *simplified*] **by** *blast*
next
have *ball-lt-lhs*: *t1* $\prec_t$ *s1* ∧ *t2* $\prec_t$ *s1*
if *rule-in*: (*s1*, *s2*) ∈ ($\bigcup$*D* ∈ {*D* ∈ *N*. *C* $\prec_c$ *D*}. *epsilon N D*)
for *s1 s2*
proof −
from *rule-in* **obtain** *D* **where**
D ∈ *N* **and** *C* $\prec_c$ *D* **and** (*s1*, *s2*) ∈ *epsilon N D*
by (*auto simp*: *rewrite-sys-def*)

have E_D-*eq*: *epsilon N D* = {(*s1*, *s2*)}
using ‹(*s1*, *s2*) ∈ *epsilon N D*› *epsilon-eq-empty-or-singleton* **by** *force*

have *l* $\prec_t$ *s1*
using ‹*C* $\prec_c$ *D*›
using *less-trm-iff-less-cls-if-lhs-epsilon*[*OF* E_D-*eq productive*]
by *metis*

with ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*› **show** *?thesis*
by (*metis reflclp-iff transpD transp-less-trm*)
qed
thus $\bigwedge$*l r*. (*l*, *r*) ∈ *rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ {*D* ∈ *N*. *C* $\prec_c$ *D*}))
⟹ *t1* $\prec_t$ *l* ∧ *t2* $\prec_t$ *l*
using *rewrite-inside-gctxt-Union-epsilon-eq*
using *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*
by *presburger*
qed

have *neg-concl1*: ¬ *upair* ‘ (*rewrite-inside-gctxt* ($\bigcup$*D* ∈ *N*. *epsilon N D*))$^\downarrow$ ⊫ *C'*
unfolding *true-cls-def Set.bex-simps*
proof (*intro ballI*)
fix *L* **assume** *L-in*: *L* ∈# *C'*
hence *L* ∈# *C*
by (*simp add*: *C-def*)

obtain *t1 t2* **where**
atm-L-eq: *atm-of L* = *Upair t1 t2*

by (*metis uprod-exhaust*)
hence *trms-of-L*: *mset-uprod* (*atm-of L*) = {#*t1*, *t2*#}
by *simp*
hence *t1* $\preceq_t$ *l* **and** *t2* $\preceq_t$ *l*
unfolding *atomize-conj*
using *less-trm-if-neg*[*OF reflclp-refl productive* ‹*L* ∈# *C*›]
using *lesseq-trm-if-pos*[*OF reflclp-refl productive* ‹*L* ∈# *C*›]
by (*metis* (*no-types*, *opaque-lifting*) *add-mset-commute sup2CI*)

have (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*))$^\downarrow$ **if** *L-def*: *L* = *Pos* (*Upair t1 t2*)
proof −
from *that* **have** (*t1*, *t2*) ∉ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^\downarrow$
using *f* ‹*L* ∈# *C'*› **by** *blast*
thus *?thesis*
using *rewrite-inside-gctxt-Union-epsilon-eq mem-join-union-iff-mem-lhs*[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
by *simp*
qed

moreover have (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*))$^\downarrow$
if *L-def*: *L* = *Neg* (*Upair t1 t2*)
proof −
from *that* **have** (*t1*, *t2*) ∈ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^\downarrow$
using *f* ‹*L* ∈# *C'*›
by (*meson true-lit-uprod-iff-true-lit-prod*(*2*) *sym-join true-cls-def true-lit-simps*(*2*))
thus *?thesis*
using *rewrite-inside-gctxt-Union-epsilon-eq*
mem-join-union-iff-mem-lhs[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
by *simp*
qed

ultimately show ¬ *upair* ‘ (*rewrite-inside-gctxt* (⋃ (*epsilon N* ‘ *N*)))$^\downarrow$ ⊫*l L*
using *atm-L-eq true-lit-uprod-iff-true-lit-prod*[*OF sym-join*] *true-lit-simps*
by (*smt* (*verit*, *ccfv-SIG*) *literal.exhaust-sel*)
qed
then show ¬ *upair* ‘ (*rewrite-inside-gctxt* (⋃ *D* ∈ *N*. *epsilon N D*))$^\downarrow$ ⊫ *C* − {#*Pos* (*Upair l r*)#}
by (*simp add*: *C-def*)
fix *D*
assume *D* ∈ *N* **and** *C* $\prec_c$ *D*
have (*l*, *r*) ∈ *rewrite-sys N D*
using *C-in* ‹(*l*, *r*) ∈ *epsilon N C*› ‹*C* $\prec_c$ *D*› *mem-rewrite-sys-if-less-cls* **by** *metis*
hence (*l*, *r*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$
by *auto*
thus *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^\downarrow$ ⊫ *C*

using *C-def* **by** *blast*

from ‹$D \in N$› **have** *rewrite-sys N D* $\subseteq$ ($\bigcup D \in N$. *epsilon N D*)
by (*simp add*: *split-Union-epsilon'*)
hence *rewrite-inside-gctxt* (*rewrite-sys N D*) $\subseteq$ *rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*)
using *rewrite-inside-gctxt-mono* **by** *metis*
hence (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\subseteq$ (*rewrite-inside-gctxt* ($\bigcup D \in N$. *epsilon N D*))$^{\downarrow}$
using *join-mono* **by** *metis*

have $\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\Vdash$ C'
unfolding *true-cls-def Set.bex-simps*
proof (*intro ballI*)
fix *L* **assume** *L-in*: $L \in\# C'$
hence $L \in\# C$
by (*simp add*: *C-def*)

obtain *t1 t2* **where**
atm-L-eq: *atm-of L* = *Upair t1 t2*
by (*metis uprod-exhaust*)
hence *trms-of-L*: *mset-uprod* (*atm-of L*) = {#*t1*, *t2*#}
by *simp*
hence *t1* $\preceq_t$ *l* **and** *t2* $\preceq_t$ *l*
unfolding *atomize-conj*
using *less-trm-if-neg*[*OF reflclp-refl productive* ‹$L \in\# C$›]
using *lesseq-trm-if-pos*[*OF reflclp-refl productive* ‹$L \in\# C$›]
by (*metis* (*no-types*, *opaque-lifting*) *add-mset-commute sup2CI*)

have (*t1*, *t2*) $\notin$ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ **if** *L-def*: *L* = *Pos* (*Upair t1 t2*)
proof –
from *that* **have** (*t1*, *t2*) $\notin$ (*rewrite-inside-gctxt* (*insert* (*l*, *r*) (*rewrite-sys N C*)))$^{\downarrow}$
using *f* ‹$L \in\# C'$› **by** *blast*
thus *?thesis*
using *rewrite-inside-gctxt-Union-epsilon-eq*
using *mem-join-union-iff-mem-lhs*[*OF* ‹*t1* $\preceq_t$ *l*› ‹*t2* $\preceq_t$ *l*›]
using ‹(*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\subseteq$ (*rewrite-inside-gctxt* ($\bigcup$ (*epsilon N* ‘ *N*)))$^{\downarrow}$› **by** *auto*
qed

moreover have (*t1*, *t2*) $\in$ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ **if** *L-def*: *L* = *Neg* (*Upair t1 t2*)
using *e*
proof (*rule contrapos-np*)
assume (*t1*, *t2*) $\notin$ (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$
hence (*t1*, *t2*) $\notin$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
using *rewrite-sys-subset-if-less-cls*[*OF* ‹$C \prec_c D$›]

by (*meson join-mono rewrite-inside-gctxt-mono subsetD*)
thus *upair* ' (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\models$ *C*
using *neg-literal-notin-imp-true-cls*[*of Upair t1 t2 C upair* ' -$^{\downarrow}$]
unfolding *uprod-mem-image-iff-prod-mem*[*OF sym-join*]
using *L-def L-in C-def*
by *simp*
qed

ultimately show ¬ *upair* ' (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\models$*l L*
using *atm-L-eq true-lit-uprod-iff-true-lit-prod*[*OF sym-join*] *true-lit-simps*
by (*smt* (*verit*, *ccfv-SIG*) *literal.exhaust-sel*)
qed
thus ¬ *upair* ' (*rewrite-inside-gctxt* (*rewrite-sys N D*))$^{\downarrow}$ $\models$ *C* − {#*Pos* (*Upair l r*)#}
by (*simp add*: *C-def*)
qed

lemma *from-neq-double-rtrancl-to-eqE*:
assumes $x \neq y$ **and** $(x, z) \in r^*$ **and** $(y, z) \in r^*$
obtains
w **where** $(x, w) \in r$ **and** $(w, z) \in r^*$ |
w **where** $(y, w) \in r$ **and** $(w, z) \in r^*$
using *assms*
by (*metis converse-rtranclE*)

lemma *ex-step-if-joinable*:
assumes *asymp R* $(x, z) \in r^*$ **and** $(y, z) \in r^*$
shows
$R^{==}\ z\ y \Longrightarrow R\ y\ x \Longrightarrow \exists w.\ (x, w) \in r \wedge (w, z) \in r^*$
$R^{==}\ z\ x \Longrightarrow R\ x\ y \Longrightarrow \exists w.\ (y, w) \in r \wedge (w, z) \in r^*$
using *assms*
by (*metis asympD converse-rtranclE reflclp-iff*)+

lemma (**in** *ground-superposition-calculus*) *trans-join-rewrite-inside-gctxt-rewrite-sys*:
trans ((*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$)
proof (*rule trans-join*)
have *wf* ((*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{-1}$)
proof (*rule wf-converse-rewrite-inside-gctxt*)
fix *s t*
assume $(s, t) \in$ *rewrite-sys N C*
then obtain *D* **where** $(s, t) \in$ *epsilon N D*
unfolding *rewrite-sys-def*
using *epsilon-filter-le-conv* **by** *auto*
thus $t \prec_t s$
by (*auto elim*: *mem-epsilonE*)
qed *auto*
thus *SN* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
by (*simp only*: *SN-iff-wf*)
next

show *WCR* (*rewrite-inside-gctxt* (*rewrite-sys N C*))
unfolding *rewrite-sys-def epsilon-filter-le-conv*
using *WCR-Union-rewrite-sys*
by (*metis* (*mono-tags*, *lifting*))
qed

lemma (**in** *ground-ordering*) *true-cls-insert-and-not-true-clsE*:
assumes
upair ‘ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ $\Vert\!\models$ *C* **and**
¬ *upair* ‘ (*rewrite-inside-gctxt R*)$^{\downarrow}$ $\Vert\!\models$ *C*
obtains $t\ t'$ **where**
Pos (*Upair* $t\ t'$) ∈# *C* **and**
$t \prec_t t'$ **and**
$(t, t') \in$ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ **and**
$(t, t') \notin$ (*rewrite-inside-gctxt R*)$^{\downarrow}$
proof −
assume *hyp*: $\bigwedge t\ t'$. *Pos* (*Upair* $t\ t'$) ∈# *C* $\Longrightarrow t \prec_t t' \Longrightarrow (t, t') \in$ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ $\Longrightarrow$
$(t, t') \notin$ (*rewrite-inside-gctxt R*)$^{\downarrow}$ $\Longrightarrow$ *thesis*

from *assms* **obtain** *L* **where**
L ∈# *C* **and**
entails-L: *upair* ‘ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$ $\Vert\!\models$*l L* **and**
doesnt-entail-L: ¬ *upair* ‘ (*rewrite-inside-gctxt R*)$^{\downarrow}$ $\Vert\!\models$*l L*
by (*meson true-cls-def*)

have *totalp-on* (*set-uprod* (*atm-of L*)) ($\prec_t$)
using *totalp-less-trm totalp-on-subset* **by** *blast*
then obtain $t\ t'$ **where** *atm-of L* = *Upair* $t\ t'$ **and** $t \preceq_t t'$
using *ex-ordered-Upair* **by** *metis*

show *?thesis*
proof (*cases L*)
case (*Pos A*)
hence *L-def*: *L* = *Pos* (*Upair* $t\ t'$)
using ‹*atm-of L* = *Upair* $t\ t'$› **by** *simp*

moreover have $(t, t') \in$ (*rewrite-inside-gctxt* (*insert r R*))$^{\downarrow}$
using *entails-L*
unfolding *L-def*
unfolding *true-lit-uprod-iff-true-lit-prod*[*OF sym-join*]
by (*simp add*: *true-lit-def*)

moreover have $(t, t') \notin$ (*rewrite-inside-gctxt R*)$^{\downarrow}$
using *doesnt-entail-L*
unfolding *L-def*
unfolding *true-lit-uprod-iff-true-lit-prod*[*OF sym-join*]
by (*simp add*: *true-lit-def*)

ultimately show *?thesis*
using *hyp* ‹$L \in\# C$› ‹$t \preceq_t t'$› **by** *auto*
next
case (*Neg A*)
hence *L-def*: $L = Neg\ (Upair\ t\ t')$
using ‹*atm-of* $L = Upair\ t\ t'$› **by** *simp*

have $(t, t') \notin (\textit{rewrite-inside-gctxt}\ (\textit{insert}\ r\ R))^{\downarrow}$
using *entails-L*
unfolding *L-def*
unfolding *true-lit-uprod-iff-true-lit-prod*[*OF sym-join*]
by (*simp add*: *true-lit-def*)

moreover have $(t, t') \in (\textit{rewrite-inside-gctxt}\ R)^{\downarrow}$
using *doesnt-entail-L*
unfolding *L-def*
unfolding *true-lit-uprod-iff-true-lit-prod*[*OF sym-join*]
by (*simp add*: *true-lit-def*)

moreover have $(\textit{rewrite-inside-gctxt}\ R)^{\downarrow} \subseteq (\textit{rewrite-inside-gctxt}\ (\textit{insert}\ r\ R))^{\downarrow}$
using *join-mono rewrite-inside-gctxt-mono*
by (*metis subset-insertI*)

ultimately have *False*
by *auto*
thus *?thesis* **..**
qed
qed

lemma (**in** *ground-superposition-calculus*) *model-preconstruction*:
fixes
N :: *'f gatom clause set* **and**
C :: *'f gatom clause*
defines
$\textit{entails} \equiv \lambda E\ C.\ \textit{upair}\ `\ (\textit{rewrite-inside-gctxt}\ E)^{\downarrow} \Vdash C$
assumes *saturated* N **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
shows
$\textit{epsilon}\ N\ C = \{\} \longleftrightarrow \textit{entails}\ (\textit{rewrite-sys}\ N\ C)\ C$
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow \textit{entails}\ (\textit{rewrite-sys}\ N\ D)\ C$
unfolding *atomize-all atomize-conj atomize-imp*
using *wfP-less-cls C-in*
proof (*induction C rule*: *wfP-induct-rule*)
case (*less C*)
note $IH = less.IH$

from ‹$\{\#\} \notin N$› ‹$C \in N$› **have** $C \neq \{\#\}$
by *metis*

define I **where**

$I = (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$

have *refl I*
by (*simp only*: *I-def refl-join*)

have *trans I*
unfolding *I-def*
using *trans-join-rewrite-inside-gctxt-rewrite-sys* **.**

have *sym I*
by (*simp only*: *I-def sym-join*)

have *compatible-with-gctxt I*
by (*simp only*: *I-def compatible-with-gctxt-join compatible-with-gctxt-rewrite-inside-gctxt*)

note *I-interp* = ‹*refl I*› ‹*trans I*› ‹*sym I*› ‹*compatible-with-gctxt I*›

have *i*: (*epsilon N C* = {}) ⟷ *entails* (*rewrite-sys N C*) *C*
proof (*rule iffI*)
show *entails* (*rewrite-sys N C*) *C* ⟹ *epsilon N C* = {}
unfolding *entails-def rewrite-sys-def*
by (*metis* (*no-types*) *empty-iff equalityI mem-epsilonE rewrite-sys-def subsetI*)
next
assume *epsilon N C* = {}

have *cond-conv*: (∃ *L*. *L* ∈# *select C* ∨ (*select C* = {#} ∧ *is-maximal-lit L C* ∧ *is-neg L*)) ⟷
(∃ *A*. *Neg A* ∈# *C* ∧ (*Neg A* ∈# *select C* ∨ *select C* = {#} ∧ *is-maximal-lit* (*Neg A*) *C*))
by (*metis* (*no-types*, *opaque-lifting*) *is-pos-def literal-order.is-maximal-in-mset-iff literal.disc*(*2*) *literal.exhaust mset-subset-eqD select-negative-lits select-subset*)

show *entails* (*rewrite-sys N C*) *C*
proof (*cases* ∃ *L*. *is-maximal-lit L* (*select C*) ∨ (*select C* = {#} ∧ *is-maximal-lit L C* ∧ *is-neg L*))
case *ex-neg-lit-sel-or-max*: *True*
hence ∃ *A*. *Neg A* ∈# *C* ∧ (*is-maximal-lit* (*Neg A*) (*select C*) ∨ *select C* = {#} ∧ *is-maximal-lit* (*Neg A*) *C*)
by (*metis is-pos-def literal.exhaust literal-order.is-maximal-in-mset-iff mset-subset-eqD select-negative-lits select-subset*)
then obtain *s s′* **where**
Neg (*Upair s s′*) ∈# *C* **and**
sel-or-max: *select C* = {#} ∧ *is-maximal-lit* (*Neg* (*Upair s s′*)) *C* ∨ *is-maximal-lit* (*Neg* (*Upair s s′*)) (*select C*)
by (*metis uprod-exhaust*)
then obtain *C′* **where**
C-def: *C* = *add-mset* (*Neg* (*Upair s s′*)) *C′*
by (*metis mset-add*)

show *?thesis*
proof (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ $\models$*l Pos* (*Upair s s'*))
  case *True*
  hence $(s, s') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
    by (*meson sym-join true-lit-simps*(*1*) *true-lit-uprod-iff-true-lit-prod*(*1*))

  have $s = s' \lor s \prec_t s' \lor s' \prec_t s$
    using *totalp-less-trm*
    by (*metis totalpD*)
  thus *?thesis*
  proof (*rule disjE*)
    assume $s = s'$
    define ι :: *'f gatom clause inference* **where**
      ι = *Infer* [*C*] *C'*

    have *ground-eq-resolution C C'*
    proof (*rule ground-eq-resolutionI*)
      show *C* = *add-mset* (*Neg* (*Upair s s'*)) *C'*
        by (*simp only*: *C-def*)
    next
      show *Neg* (*Upair s s'*) = *Neg* (*Upair s s*)
        by (*simp only*: ‹$s = s'$›)
    next
      show *select C* = {#} ∧ *is-maximal-lit* (*s* !≈ *s'*) *C* ∨ *is-maximal-lit* (*s* !≈ *s'*) (*select C*)
        using *sel-or-max* **.**
    qed *simp*
    hence $\iota \in$ *G-Inf*
      by (*auto simp only*: *ι-def G-Inf-def*)

    moreover have $\bigwedge t.\ t \in$ *set* (*prems-of* ι) $\Longrightarrow t \in N$
      using ‹$C \in N$›
      by (*simp add*: *ι-def*)

    ultimately have $\iota \in$ *Inf-from N*
      by (*auto simp*: *Inf-from-def*)
    hence $\iota \in$ *Red-I N*
      using ‹*saturated N*›
      by (*auto simp*: *saturated-def*)
    then obtain *DD* **where**
      DD-subset: $DD \subseteq N$ **and**
      finite DD **and**
      DD-entails-C': *G-entails DD* {*C'*} **and**
      ball-DD-lt-C: $\forall D \in DD.\ D \prec_c C$
      unfolding *Red-I-def redundant-infer-def*
      by (*auto simp*: *ι-def*)

    moreover have $\forall D \in DD.$ *entails* (*rewrite-sys N C*) *D*

using *IH*[*THEN conjunct2*, *rule-format*, *of* - *C*]
using ‹$C \in N$› *DD-subset ball-DD-lt-C*
by *blast*

ultimately have *entails* (*rewrite-sys N C*) C'
using *I-interp DD-entails-C*$'$
unfolding *entails-def G-entails-def*
by (*simp add*: *I-def true-clss-def*)
then show *entails* (*rewrite-sys N C*) *C*
using *C-def entails-def* **by** *simp*
next
from ‹$(s, s') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$› **obtain** *u* **where**
s-u: $(s, u) \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))* **and**
s'*-u*: $(s', u) \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))*
by *auto*
moreover hence $u \preceq_t s$ **and** $u \preceq_t s'$
using *rhs-lesseq-trm-lhs-if-mem-rtrancl-rewrite-inside-gctxt-rewrite-sys*
by *simp-all*

moreover assume $s \prec_t s' \vee s' \prec_t s$

ultimately obtain u_0 **where**
$s' \prec_t s \Longrightarrow (s, u_0)$: *rewrite-inside-gctxt* (*rewrite-sys N C*)
$s \prec_t s' \Longrightarrow (s', u_0)$: *rewrite-inside-gctxt* (*rewrite-sys N C*) **and**
(u_0, u) : (*rewrite-inside-gctxt* (*rewrite-sys N C*))*
using *ex-step-if-joinable*[*OF* - *s-u* s'*-u*]
by (*metis asympD asymp-less-trm*)
then obtain *ctxt t* t' **where**
s-eq-if: $s' \prec_t s \Longrightarrow s = ctxt\langle t\rangle_G$ **and**
s'*-eq-if*: $s \prec_t s' \Longrightarrow s' = ctxt\langle t\rangle_G$ **and**
$u_0 = ctxt\langle t'\rangle_G$ **and**
$(t, t') \in$ *rewrite-sys N C*
by (*smt* (*verit*) *Pair-inject* ‹$s \prec_t s' \vee s' \prec_t s$› *asympD asymp-less-trm*
mem-Collect-eq
rewrite-inside-gctxt-def)
then obtain *D* **where**
$(t, t') \in$ *epsilon N D* **and** $D \in N$ **and** $D \prec_c C$
unfolding *rewrite-sys-def epsilon-filter-le-conv* **by** *auto*
then obtain D' **where**
D-def: $D =$ *add-mset* (*Pos* (*Upair t* t')) D' **and**
sel-D: *select* $D = \{\#\}$ **and**
max-t-t$'$: *is-strictly-maximal-lit* (*Pos* (*Upair t* t')) *D* **and**
$t' \prec_t t$
by (*elim mem-epsilonE*) *fast*

have *superI*: *ground-neg-superposition D C* (*add-mset* (*Neg* (*Upair* $s_1\langle t'\rangle_G$
$s_1{}'$)) $(C' + D')$)
if $\{s, s'\} = \{s_1\langle t\rangle_G, s_1{}'\}$ **and** $s_1{}' \prec_t s_1\langle t\rangle_G$
for s_1 $s_1{}'$

proof (*rule ground-neg-superpositionI*)
show C = *add-mset* (*Neg* (*Upair s s'*)) C'
by (*simp only*: *C-def*)
next
show D = *add-mset* (*Pos* (*Upair t t'*)) D'
by (*simp only*: *D-def*)
next
show $D \prec_c C$
using ‹$D \prec_c C$› **.**
next
show *select C* = {#} ∧ *is-maximal-lit* (*Neg* (*Upair s s'*)) C ∨ *is-maximal-lit* (s !≈ s') (*select C*)
using *sel-or-max* **.**
next
show *select D* = {#}
using *sel-D* **.**
next
show *is-strictly-maximal-lit* (*Pos* (*Upair t t'*)) D
using *max-t-t'* **.**
next
show $t' \prec_t t$
using ‹$t' \prec_t t$› **.**
next
from *that*(*1*) **show** *Neg* (*Upair s s'*) = *Neg* (*Upair* $s_1\langle t\rangle_G$ s_1')
by *fastforce*
next
from *that*(*2*) **show** $s_1' \prec_t s_1\langle t\rangle_G$ **.**
qed *simp-all*

have *ground-neg-superposition D C* (*add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ s')) ($C' + D'$))
if ‹$s' \prec_t s$›
proof (*rule superI*)
from *that* **show** $\{s, s'\} = \{ctxt\langle t\rangle_G, s'\}$
using *s-eq-if* **by** *simp*
next
from *that* **show** $s' \prec_t ctxt\langle t\rangle_G$
using *s-eq-if* **by** *simp*
qed

moreover have *ground-neg-superposition D C* (*add-mset* (*Neg* (*Upair* $ctxt\langle t'\rangle_G$ s)) ($C' + D'$))
if ‹$s \prec_t s'$›
proof (*rule superI*)
from *that* **show** $\{s, s'\} = \{ctxt\langle t\rangle_G, s\}$
using *s'-eq-if* **by** *auto*
next
from *that* **show** $s \prec_t ctxt\langle t\rangle_G$
using *s'-eq-if* **by** *simp*

qed

ultimately obtain *CD* **where**
super: *ground-neg-superposition D C CD* **and**
CD-eq1: $s' \prec_t s \Longrightarrow CD = add\text{-}mset\ (Neg\ (Upair\ ctxt\langle t'\rangle_G\ s'))\ (C' + D')$ **and**
CD-eq2: $s \prec_t s' \Longrightarrow CD = add\text{-}mset\ (Neg\ (Upair\ ctxt\langle t'\rangle_G\ s))\ (C' + D')$
using ‹$s \prec_t s' \vee s' \prec_t s$› *s′-eq-if s-eq-if* **by** *metis*

define ι :: *′f gatom clause inference* **where**
$\iota = Infer\ [D, C]\ CD$

have $\iota \in G\text{-}Inf$
using *ground-superposition-if-ground-neg-superposition*[*OF super*]
by (*auto simp only*: *ι-def G-Inf-def*)

moreover have $\bigwedge t.\ t \in set\ (prems\text{-}of\ \iota) \Longrightarrow t \in N$
using ‹$C \in N$› ‹$D \in N$›
by (*auto simp add*: *ι-def*)

ultimately have $\iota \in Inf\text{-}from\ N$
by (*auto simp*: *Inf-from-def*)
hence $\iota \in Red\text{-}I\ N$
using ‹*saturated N*›
by (*auto simp*: *saturated-def*)
then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-CD: *G-entails* (*insert D DD*) {*CD*} **and**
ball-DD-lt-C: $\forall D{\in}DD.\ D \prec_c C$
unfolding *Red-I-def redundant-infer-def mem-Collect-eq*
by (*auto simp*: *ι-def*)

moreover have $\forall D{\in}$ *insert D DD*. *entails* (*rewrite-sys N C*) *D*
using *IH*[*THEN conjunct2*, *rule-format*, *of - C*]
using ‹$C \in N$› ‹$D \in N$› ‹$D \prec_c C$› *DD-subset ball-DD-lt-C*
by (*metis in-mono insert-iff*)

ultimately have *entails* (*rewrite-sys N C*) *CD*
using *I-interp DD-entails-CD*
unfolding *entails-def G-entails-def*
by (*simp add*: *I-def true-clss-def*)

moreover have ¬ *entails* (*rewrite-sys N C*) *D′*
unfolding *entails-def*
using *false-cls-if-productive-epsilon*(*2*)[*OF* - ‹$C \in N$› ‹$D \prec_c C$›]
by (*metis D-def* ‹$(t, t') \in$ *epsilon N D*› *add-mset-remove-trivial empty-iff epsilon-eq-empty-or-singleton singletonD*)

moreover have ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫*l*
(*Neg* (*Upair* *ctxt*⟨*t*′⟩$_G$ *s*′))
if $s' \prec_t s$
using ‹(u_0, *u*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))*› ‹u_0 = *ctxt*⟨*t*′⟩$_G$›
s′*-u* **by** *blast*

moreover have ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫*l*
(*Neg* (*Upair* *ctxt*⟨*t*′⟩$_G$ *s*))
if $s \prec_t s'$
using ‹(u_0, *u*) ∈ (*rewrite-inside-gctxt* (*rewrite-sys N C*))*› ‹u_0 = *ctxt*⟨*t*′⟩$_G$›
s-u **by** *blast*

ultimately show *entails* (*rewrite-sys N C*) *C*
unfolding *entails-def C-def*
using ‹$s \prec_t s' \vee s' \prec_t s$› *CD-eq1 CD-eq2* **by** *fast*
qed
next
case *False*
thus *?thesis*
using ‹*Neg* (*Upair s s*′) ∈# *C*›
by (*auto simp add*: *entails-def true-cls-def*)
qed
next
case *False*
hence *select C* = {#}
using *literal-order.ex-maximal-in-mset* **by** *blast*

from *False* **obtain** *A* **where** *Pos-A-in*: *Pos A* ∈# *C* **and** *max-Pos-A*:
is-maximal-lit (*Pos A*) *C*
using ‹*select C* = {#}› *literal-order.ex-maximal-in-mset*[*OF* ‹*C* ≠ {#}›]
by (*metis is-pos-def literal-order.is-maximal-in-mset-iff*)
then obtain *C*′ **where** *C-def*: *C* = *add-mset* (*Pos A*) *C*′
by (*meson mset-add*)

have *totalp-on* (*set-uprod A*) ($\prec_t$)
using *totalp-less-trm totalp-on-subset* **by** *blast*
then obtain *s s*′ **where** *A-def*: *A* = *Upair s s*′ **and** $s' \preceq_t s$
using *ex-ordered-Upair*[*of A* ($\prec_t$)] **by** *fastforce*

show *?thesis*
proof (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ *C*′ ∨ *s* = *s*′)
case *True*
then show *?thesis*
using ‹*epsilon N C* = {}›
using *A-def C-def entails-def* **by** *blast*
next
case *False*

from *False* **have** ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ *C*′

by *simp*

from *False* **have** $s' \prec_t s$
using ‹$s' \preceq_t s$› *asymp-less-trm*[*THEN asympD*] **by** *auto*

then show *?thesis*
proof (*cases is-strictly-maximal-lit* (*Pos A*) *C*)
case *strictly-maximal*: *True*
show *?thesis*
proof (*cases s* ∈ *NF* (*rewrite-inside-gctxt* (*rewrite-sys N C*)))
case *s-irreducible*: *True*
hence *e-or-f-doesnt-hold*: *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
⊫ *C* ∨
upair ‘ (*rewrite-inside-gctxt* (*insert* (*s*, *s*′) (*rewrite-sys N C*)))$^{\downarrow}$ ⊫ *C*′
using ‹*epsilon N C* = {}›[*unfolded epsilon.simps*[*of N C*]]
using ‹*C* ∈ *N*› *C-def* ‹*select C* = {#}› *strictly-maximal* ‹$s' \prec_t s$›
unfolding *A-def rewrite-sys-def*
by (*smt* (*verit*, *best*) *Collect-empty-eq*)
show *?thesis*
proof (*cases upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$ ⊫ *C*)
case *e-doesnt-hold*: *True*
thus *?thesis*
by (*simp add*: *entails-def*)
next
case *e-holds*: *False*
hence *R-C-doesnt-entail-C*′: ¬ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys*
N C))$^{\downarrow}$ ⊫ *C*′
unfolding *C-def* **by** *simp*
show *?thesis*
proof (*cases upair* ‘ (*rewrite-inside-gctxt* (*insert* (*s*, *s*′) (*rewrite-sys N*
C)))$^{\downarrow}$ ⊫ *C*′)
case *f-doesnt-hold*: *True*
then obtain *C*′′ *t t*′ **where** *C*′*-def*: *C*′ = *add-mset* (*Pos* (*Upair t*
t′)) *C*′′ **and**
$t' \prec_t t$ **and**
(*t*, *t*′) ∈ (*rewrite-inside-gctxt* (*insert* (*s*, *s*′) (*rewrite-sys N C*)))$^{\downarrow}$ **and**
(*t*, *t*′) ∉ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
using *f-doesnt-hold R-C-doesnt-entail-C*′
using *true-cls-insert-and-not-true-clsE*
by (*metis insert-DiffM join-sym Upair-sym*)

have *Pos* (*Upair t t*′) $\prec_l$ *Pos* (*Upair s s*′)
using *strictly-maximal*
by (*simp add*: *A-def C*′*-def C-def literal-order.is-greatest-in-mset-iff*)

have ¬ ($t \prec_t s$)
proof (*rule notI*)
assume $t \prec_t s$

have $(t,\ t') \in (\textit{rewrite-inside-gctxt}\ (\textit{insert}\ (s,\ s')\ (\textit{rewrite-sys}\ N\ C)))^{\downarrow} \longleftrightarrow$
$(t,\ t') \in (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$
unfolding *rewrite-inside-gctxt-union*[*of* {(*s*, *s′*)} *rewrite-sys N C*, *simplified*]
proof (*rule mem-join-union-iff-mem-join-rhs′*)
show $\bigwedge t1\ t2.\ (t1,\ t2) \in \textit{rewrite-inside-gctxt}\ \{(s,\ s')\} \Longrightarrow t \prec_t t1 \wedge t' \prec_t t1$
using ‹$t \prec_t s$› ‹$t' \prec_t t$›
by (*smt* (*verit*, *ccfv-threshold*) *fst-conv singletonD less-trm-const-lhs-if-mem-rewrite-inside-gctxt transpD transp-less-trm*)
next
show $\bigwedge t1\ t2.\ (t1,\ t2) \in \textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C) \Longrightarrow t2 \prec_t t1$
using *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys* **by** *force*
qed
thus *False*
using ‹$(t,\ t') \in (\textit{rewrite-inside-gctxt}\ (\textit{insert}\ (s,\ s')\ (\textit{rewrite-sys}\ N\ C)))^{\downarrow}$›
using ‹$(t,\ t') \notin (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$›
by *metis*
qed

moreover have $\neg\ (s \prec_t t)$
proof (*rule notI*)
assume $s \prec_t t$
hence *multp* $(\prec_t)$ {#*s*, *s′*#} {#*t*, *t′*#}
using ‹$s' \prec_t s$› ‹$t' \prec_t t$›
using *one-step-implies-multp*[*of* - - - {#}, *simplified*]
by (*metis* (*mono-tags*, *opaque-lifting*) *empty-not-add-mset insert-iff set-mset-add-mset-insert set-mset-empty singletonD transpD transp-less-trm*)
hence *Pos* (*Upair s s′*) $\prec_l$ *Pos* (*Upair t t′*)
by (*simp add*: *less-lit-def*)
thus *False*
using ‹$t \approx t' \prec_l s \approx s'$› **by** *order*
qed

ultimately have $t = s$
by *order*
hence $t' \prec_t s'$
using ‹$t' \prec_t t$› ‹$s' \prec_t s$›
using ‹*Pos* (*Upair t t′*) $\prec_l$ *Pos* (*Upair s s′*)›
unfolding *less-lit-def*
by (*simp add*: *multp-cancel-add-mset transp-less-trm*)

obtain t'' **where**

$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*)) **and**
$(t'', t') \in$ (*rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*)))$^{\downarrow}$
using ‹$(t, t') \in$ (*rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*)))$^{\downarrow}$›[*THEN joinD*]
using *ex-step-if-joinable*[*OF asymp-less-trm - - -* ‹$t' \prec_t t$›]
by (*smt* (*verit, ccfv-threshold*) ‹$t = s$› *converse-rtranclE insertCI joinI-right*
join-sym r-into-rtrancl mem-rewrite-inside-gctxt-if-mem-rewrite-rules rtrancl-join-join)

have $t'' \prec_t t$
proof (*rule predicate-holds-of-mem-rewrite-inside-gctxt*[*of - - -* $\lambda x\ y.\ y \prec_t x$])
show $(t, t'') \in$ *rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*))
using ‹$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*))› **.**
next
show $\bigwedge t1\ t2.\ (t1, t2) \in$ *insert* (s, s') (*rewrite-sys N C*) $\Longrightarrow t2 \prec_t t1$
by (*metis* ‹$s' \prec_t s$› *insert-iff old.prod.inject rhs-lt-lhs-if-mem-rewrite-sys*)
next
show $\bigwedge t1\ t2\ ctxt\ \sigma.\ (t1, t2) \in$ *insert* (s, s') (*rewrite-sys N C*) $\Longrightarrow$
$t2 \prec_t t1 \Longrightarrow ctxt\langle t2\rangle_G \prec_t ctxt\langle t1\rangle_G$
by (*simp only*: *less-trm-compatible-with-gctxt*)
qed

have $(t, t'') \in$ *rewrite-inside-gctxt* $\{(s, s')\}$
using ‹$(t, t'') \in$ *rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*))›
using ‹$t = s$› *s-irreducible mem-rewrite-step-union-NF*
using *rewrite-inside-gctxt-insert* **by** *blast*
hence $\exists\, ctxt.\ s = ctxt\langle s\rangle_G \wedge t'' = ctxt\langle s'\rangle_G$
by (*simp add*: ‹$t = s$› *rewrite-inside-gctxt-def*)
hence $t'' = s'$
by (*metis gctxt-ident-iff-eq-GHole*)

moreover have $(t'', t') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
proof (*rule mem-join-union-iff-mem-join-rhs'*[*THEN iffD1*])
show $(t'', t') \in$ (*rewrite-inside-gctxt* $\{(s, s')\} \cup$
rewrite-inside-gctxt (*rewrite-sys N C*))$^{\downarrow}$
using ‹$(t'', t') \in$ (*rewrite-inside-gctxt* (*insert* (s, s') (*rewrite-sys N C*)))$^{\downarrow}$›
using *rewrite-inside-gctxt-union*[*of* {-}, *simplified*] **by** *metis*
next
show $\bigwedge t1\ t2.\ (t1, t2) \in$ *rewrite-inside-gctxt* (*rewrite-sys N C*) $\Longrightarrow t2 \prec_t t1$
using *rhs-less-trm-lhs-if-mem-rewrite-inside-gctxt-rewrite-sys* **.**
next
show $\bigwedge t1\ t2.\ (t1, t2) \in$ *rewrite-inside-gctxt* $\{(s, s')\} \Longrightarrow t'' \prec_t t1$

$\wedge$ $t' \prec_t$ *t1*
    using ‹$t' \prec_t t$› ‹$t'' \prec_t t$›
    unfolding ‹$t = s$›
    using *less-trm-const-lhs-if-mem-rewrite-inside-gctxt* **by** *fastforce*
  qed

  ultimately have $(s', t') \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^{\downarrow}$
    by *simp*

  let *?concl* = *add-mset* (*Neg* (*Upair* s' t')) (*add-mset* (*Pos* (*Upair t* t')) C'')

  define ι :: *'f gatom clause inference* **where**
    ι = *Infer* [*C*] *?concl*

  have *eq-fact*: *ground-eq-factoring C ?concl*
  proof (*rule ground-eq-factoringI*)
    show *C* = *add-mset* (*Pos* (*Upair s* s')) (*add-mset* (*Pos* (*Upair t* t')) C'')
      by (*simp add*: *C-def* C'*-def A-def*)
  next
    show *select C* = {#}
      using ‹*select C* = {#}› **.**
  next
    show *is-maximal-lit* (*Pos* (*Upair s* s')) *C*
      by (*metis A-def max-Pos-A*)
  next
    show $s' \prec_t s$
      using ‹$s' \prec_t s$› **.**
  next
    show *Pos* (*Upair t* t') = *Pos* (*Upair s* t')
      unfolding ‹$t = s$› **..**
  next
    show *add-mset* (*Neg* (*Upair* s' t')) (*add-mset* (*Pos* (*Upair t* t')) C'') =
      add-mset (*Neg* (*Upair* s' t')) (*add-mset* (*Pos* (*Upair s* t')) C'')
      by (*auto simp add*: ‹$t = s$›)
  qed *simp-all*
  hence $\iota \in$ *G-Inf*
    by (*auto simp*: ι*-def G-Inf-def*)

  moreover have $\bigwedge t.\ t \in$ *set* (*prems-of* ι) $\Longrightarrow t \in N$
    using ‹$C \in N$›
    by (*auto simp add*: ι*-def*)

  ultimately have $\iota \in$ *Inf-from N*
    by (*auto simp*: *Inf-from-def*)
  hence $\iota \in$ *Red-I N*
    using ‹*saturated N*›

by (*auto simp*: *saturated-def*)
then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-C': *G-entails DD* {*?concl*} **and**
ball-DD-lt-C: $\forall D \in DD.\ D \prec_c C$
unfolding *Red-I-def redundant-infer-def*
by (*auto simp*: *ι-def*)

have $\forall D \in DD.$ *entails* (*rewrite-sys N C*) *D*
using *IH*[*THEN conjunct2*, *rule-format*, *of - C*]
using ‹$C \in N$› *DD-subset ball-DD-lt-C*
by *blast*
hence *entails* (*rewrite-sys N C*) *?concl*
unfolding *entails-def I-def*[*symmetric*]
using *DD-entails-C'*[*unfolded G-entails-def*]
using *I-interp*
by (*simp add*: *true-clss-def*)
thus *entails* (*rewrite-sys N C*) *C*
unfolding *entails-def I-def*[*symmetric*]
unfolding *C-def C'-def A-def*
using *I-def* ‹$(s', t') \in (\textit{rewrite-inside-gctxt}\ (\textit{rewrite-sys}\ N\ C))^{\downarrow}$› **by**
blast
next
case *f-holds*: *False*
hence *False*
using *e-or-f-doesnt-hold e-holds* **by** *metis*
thus *?thesis* **..**
qed
qed
next
case *s-reducible*: *False*
hence $\exists ss.$ (*s*, *ss*) $\in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
unfolding *NF-def* **by** *auto*
then obtain *ctxt t t' D* **where**
$D \in N$ **and**
$D \prec_c C$ **and**
$(t, t') \in$ *epsilon N D* **and**
$s = ctxt\langle t\rangle_G$
using *epsilon-filter-le-conv*
by (*auto simp*: *rewrite-inside-gctxt-def rewrite-sys-def*)

obtain *D'* **where**
D-def: *D* = *add-mset* (*Pos* (*Upair t t'*)) *D'* **and**
select D = {#} **and**
max-t-t': *is-strictly-maximal-lit* ($t \approx t'$) *D* **and**
$t' \prec_t t$
using ‹$(t, t') \in$ *epsilon N D*›
by (*elim mem-epsilonE*) *simp*

let *?concl* = *add-mset* (*Pos* (*Upair* $ctxt\langle t'\rangle_G$ *s'*)) ($C' + D'$)

define ι :: *'f gatom clause inference* **where**
  ι = *Infer* [*D*, *C*] *?concl*

have *super*: *ground-pos-superposition D C ?concl*
proof (*rule ground-pos-superpositionI*)
  show *C* = *add-mset* (*Pos* (*Upair s s'*)) *C'*
    by (*simp only*: *C-def A-def*)
next
  show *D* = *add-mset* (*Pos* (*Upair t t'*)) *D'*
    by (*simp only*: *D-def*)
next
  show $D \prec_c C$
    using ‹$D \prec_c C$› **.**
next
  show *select D* = {#}
    using ‹*select D* = {#}› **.**
next
  show *select C* = {#}
    using ‹*select C* = {#}› **.**
next
  show *is-strictly-maximal-lit* ($s \approx s'$) *C*
    using *A-def strictly-maximal* **by** *simp*
next
  show *is-strictly-maximal-lit* ($t \approx t'$) *D*
    using *max-t-t'* **.**
next
  show $t' \prec_t t$
    using ‹$t' \prec_t t$› **.**
next
  show *Pos* (*Upair s s'*) = *Pos* (*Upair* $ctxt\langle t\rangle_G$ *s'*)
    by (*simp only*: ‹$s = ctxt\langle t\rangle_G$›)
next
  show $s' \prec_t ctxt\langle t\rangle_G$
    using ‹$s' \prec_t s$›
    unfolding ‹$s = ctxt\langle t\rangle_G$› **.**
qed *simp-all*
hence $\iota \in$ *G-Inf*
  using *ground-superposition-if-ground-pos-superposition*
  by (*auto simp*: ι*-def G-Inf-def*)

moreover have $\bigwedge t.\ t \in$ *set* (*prems-of* ι) $\Longrightarrow t \in N$
  using ‹$C \in N$› ‹$D \in N$›
  by (*auto simp add*: ι*-def*)

ultimately have $\iota \in$ *Inf-from N*
  by (*auto simp only*: *Inf-from-def*)

hence $\iota \in$ *Red-I N*
using ‹*saturated N*›
by (*auto simp only*: *saturated-def*)
then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-concl: *G-entails* (*insert D DD*) {*?concl*} **and**
ball-DD-lt-C: $\forall D \in DD.\ D \prec_c C$
unfolding *Red-I-def redundant-infer-def mem-Collect-eq*
by (*auto simp*: ι*-def*)

moreover have $\forall D \in$ *insert D DD*. *entails* (*rewrite-sys N C*) *D*
using *IH*[*THEN conjunct2*, *rule-format*, *of - C*]
using ‹$C \in N$› ‹$D \in N$› ‹$D \prec_c C$› *DD-subset ball-DD-lt-C*
by (*metis in-mono insert-iff*)

ultimately have *entails* (*rewrite-sys N C*) *?concl*
using *I-interp DD-entails-concl*
unfolding *entails-def G-entails-def*
by (*simp add*: *I-def true-clss-def*)

moreover have $\neg$ *entails* (*rewrite-sys N C*) D'
unfolding *entails-def*
using *false-cls-if-productive-epsilon*(*2*)[*OF* - ‹$C \in N$› ‹$D \prec_c C$›]
by (*metis D-def* ‹$(t, t') \in$ *epsilon N D*› *add-mset-remove-trivial empty-iff epsilon-eq-empty-or-singleton singletonD*)

ultimately have *entails* (*rewrite-sys N C*) {#*Pos* (*Upair ctxt*$\langle t' \rangle_G$ s')#}
unfolding *entails-def*
using ‹$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$ $\Vdash$ C'›
by *fastforce*

hence (*ctxt*$\langle t' \rangle_G$, s') $\in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
by (*simp add*: *entails-def true-cls-def uprod-mem-image-iff-prod-mem*[*OF sym-join*])

moreover have (*ctxt*$\langle t \rangle_G$, *ctxt*$\langle t' \rangle_G$) $\in$ *rewrite-inside-gctxt* (*rewrite-sys N C*)
using ‹$(t, t') \in$ *epsilon N D*› ‹$D \in N$› ‹$D \prec_c C$› *rewrite-sys-def epsilon-filter-le-conv*
by (*auto simp*: *rewrite-inside-gctxt-def*)

ultimately have (*ctxt*$\langle t \rangle_G$, s') $\in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))$^\downarrow$
using *r-into-rtrancl rtrancl-join-join* **by** *metis*

hence *entails* (*rewrite-sys N C*) {#*Pos* (*Upair ctxt*$\langle t \rangle_G$ s')#}
unfolding *entails-def true-cls-def* **by** *auto*

```
      thus ?thesis
        using A-def C-def ‹s = ctxt⟨t⟩G› entails-def by fastforce
    qed
  next
    case False
    hence 2 ≤ count C (Pos A)
      using max-Pos-A
   by (metis literal-order.count-ge-2-if-maximal-in-mset-and-not-greatest-in-mset)
   then obtain C' where C-def: C = add-mset (Pos A) (add-mset (Pos A)
C')
      using two-le-countE by metis

    define ι :: 'f gatom clause inference where
      ι = Infer [C] (add-mset (Pos (Upair s s')) (add-mset (Neg (Upair s' s'))
C'))

   let ?concl = add-mset (Pos (Upair s s')) (add-mset (Neg (Upair s' s')) C')

    have eq-fact: ground-eq-factoring C ?concl
    proof (rule ground-eq-factoringI)
      show C = add-mset (Pos A) (add-mset (Pos A) C')
        by (simp add: C-def)
    next
      show Pos A = Pos (Upair s s')
        by (simp add: A-def)
    next
      show Pos A = Pos (Upair s s')
        by (simp add: A-def)
    next
      show select C = {#}
        using ‹select C = {#}› .
    next
      show is-maximal-lit (Pos A) C
        using max-Pos-A .
    next
      show s' ≺t s
        using ‹s' ≺t s› .
    qed simp-all
    hence ι ∈ G-Inf
      by (auto simp: ι-def G-Inf-def)

    moreover have ⋀t. t ∈ set (prems-of ι) ⟹ t ∈ N
      using ‹C ∈ N›
      by (auto simp add: ι-def)

    ultimately have ι ∈ Inf-from N
      by (auto simp: Inf-from-def)
    hence ι ∈ Red-I N
      using ‹saturated N›
```

by (*auto simp*: *saturated-def*)
then obtain *DD* **where**
DD-subset: $DD \subseteq N$ **and**
finite DD **and**
DD-entails-concl: *G-entails DD* {*?concl*} **and**
ball-DD-lt-C: $\forall D \in DD.\ D \prec_c C$
unfolding *Red-I-def redundant-infer-def mem-Collect-eq*
by (*auto simp*: *ι-def*)

moreover have $\forall D \in DD.$ *entails* (*rewrite-sys N C*) *D*
using *IH*[*THEN conjunct2*, *rule-format*, *of - C*]
using ‹$C \in N$› *DD-subset ball-DD-lt-C*
by *blast*

ultimately have *entails* (*rewrite-sys N C*) *?concl*
using *I-interp DD-entails-concl*
unfolding *entails-def G-entails-def*
by (*simp add*: *I-def true-clss-def*)
then show *?thesis*
by (*simp add*: *entails-def A-def C-def joinI-right pair-imageI*)
qed
qed
qed
qed

moreover have *iib*: *entails* (*rewrite-sys N D*) *C* **if** $D \in N$ **and** $C \prec_c D$ **for** *D*
using *epsilon-eq-empty-or-singleton*[*of N C*, *folded*]
proof (*elim disjE exE*)
assume *epsilon N C* = {}
hence *entails* (*rewrite-sys N C*) *C*
unfolding *i* **by** *simp*
thus *?thesis*
using *lift-entailment-to-Union*(*2*)[*OF* ‹$C \in N$› *- that*]
by (*simp only*: *entails-def*)
next
fix *l r* **assume** *epsilon N C* = {(*l*, *r*)}
thus *?thesis*
using *true-cls-if-productive-epsilon*(*2*)[*OF* ‹*epsilon N C* = {(*l*, *r*)}› *that*]
by (*simp only*: *entails-def*)
qed

ultimately show *?case*
by *metis*
qed

lemma (**in** *ground-superposition-calculus*) *model-construction*:
fixes
N :: *'f gatom clause set* **and**
C :: *'f gatom clause*

defines
  entails ≡ $\lambda E\ C.$ *upair* ‘ (*rewrite-inside-gctxt* $E)^{\downarrow} \models C$
assumes *saturated* N **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
shows *entails* ($\bigcup D \in N.$ *epsilon* $N\ D$) C
using *epsilon-eq-empty-or-singleton*[*of* $N\ C$]
proof (*elim disjE exE*)
  assume *epsilon* $N\ C = \{\}$
  hence *entails* (*rewrite-sys* $N\ C$) C
    using *model-preconstruction*(*1*)[*OF assms*(*2*,*3*,*4*)] **by** (*metis entails-def*)
  thus *?thesis*
    using *lift-entailment-to-Union*(*1*)[*OF* ‹$C \in N$›]
    by (*simp only*: *entails-def*)
next
  fix $l\ r$ **assume** *epsilon* $N\ C = \{(l, r)\}$
  thus *?thesis*
    using *true-cls-if-productive-epsilon*(*1*)[*OF* ‹*epsilon* $N\ C = \{(l, r)\}$›]
    by (*simp only*: *entails-def*)
qed

1.5 Static Refutational Completeness

lemma (**in** *ground-superposition-calculus*) *statically-complete*:
  fixes N :: *'f gatom clause set*
  assumes *saturated* N **and** *G-entails* N $\{\{\#\}\}$
  shows $\{\#\} \in N$
  using ‹*G-entails* N $\{\{\#\}\}$›
proof (*rule contrapos-pp*)
  assume $\{\#\} \notin N$

  define I :: *'f gterm rel* **where**
    I = (*rewrite-inside-gctxt* ($\bigcup D \in N.$ *epsilon* $N\ D$))$^{\downarrow}$

  show ¬ *G-entails* N *G-Bot*
    unfolding *G-entails-def not-all not-imp*
  proof (*intro exI conjI*)
    show *refl* I
      by (*simp only*: *I-def refl-join*)
  next
    show *trans* I
      unfolding *I-def*
    proof (*rule trans-join*)
      have *wf* ((*rewrite-inside-gctxt* ($\bigcup D \in N.$ *epsilon* $N\ D$))$^{-1}$)
      proof (*rule wf-converse-rewrite-inside-gctxt*)
        fix $s\ t$
        assume $(s, t) \in (\bigcup D \in N.$ *epsilon* $N\ D)$
        then obtain C **where** $C \in N$ $(s, t) \in$ *epsilon* $N\ C$
          by *auto*
        thus $t \prec_t s$
          by (*auto elim*: *mem-epsilonE*)

```
      qed auto
      thus SN (rewrite-inside-gctxt (⋃D ∈ N. epsilon N D))
        unfolding SN-iff-wf .
    next
      show WCR (rewrite-inside-gctxt (⋃D ∈ N. epsilon N D))
        using WCR-Union-rewrite-sys .
    qed
  next
    show sym I
      by (simp only: I-def sym-join)
  next
    show compatible-with-gctxt I
      unfolding I-def
    by (simp only: I-def compatible-with-gctxt-join compatible-with-gctxt-rewrite-inside-gctxt)
  next
    show upair ‘ I ⊫s N
      unfolding I-def
      using model-construction[OF ‹saturated N› ‹{#} ∉ N›]
      by (simp add: true-clss-def)
  next
    show ¬ upair ‘ I ⊫s G-Bot
      by simp
  qed
qed

sublocale ground-superposition-calculus ⊆ statically-complete-calculus where
  Bot = G-Bot and
  Inf = G-Inf and
  entails = G-entails and
  Red-I = Red-I and
  Red-F = Red-F
proof unfold-locales
  fix B :: 'f gatom clause and N :: 'f gatom clause set
  assume B ∈ G-Bot and saturated N
  hence B = {#}
    by simp

  assume G-entails N {B}
  hence {#} ∈ N
    unfolding ‹B = {#}›
    using statically-complete[OF ‹saturated N›] by argo
  thus ∃B'∈G-Bot. B' ∈ N
    by auto
qed

end
theory Variable-Substitution
  imports
    Abstract-Substitution.Substitution
```

HOL−Library.FSet
HOL−Library.Multiset
begin

locale *finite-set* =
fixes *set* :: *'b* ⇒ *'a set*
assumes *finite-set* [*simp*]: ⋀*b*. *finite* (*set b*)
begin

abbreviation *finite-set* :: *'b* ⇒ *'a fset* **where**
finite-set b ≡ *Abs-fset* (*set b*)

lemma *finite-set'*: *set b* ∈ {*A*. *finite A*}
by *simp*

lemma *fset-finite-set* [*simp*]: *fset* (*finite-set b*) = *set b*
using *Abs-fset-inverse*[*OF finite-set'*]**.**

end

locale *variable-substitution* = *substitution* - - *subst* λ*a*. *vars a* = {}
for
subst :: *'expression* ⇒ (*'variable* ⇒ *'base-expression*) ⇒ *'expression* (**infixl** · *70*)
and
vars :: *'expression* ⇒ *'variable set* +
assumes
subst-eq: ⋀*a σ τ*. (⋀*x*. *x* ∈ (*vars a*) ⟹ *σ x* = *τ x*) ⟹ *a* · *σ* = *a* · *τ*
begin

abbreviation *is-ground* **where** *is-ground a* ≡ *vars a* = {}

definition *vars-set* :: *'expression set* ⇒ *'variable set* **where**
vars-set expressions ≡ ⋃ *expression* ∈ *expressions*. *vars expression*

lemma *subst-reduntant-upd* [*simp*]:
assumes *var* ∉ *vars a*
shows *a* · *σ*(*var* := *update*) = *a* · *σ*
using *assms subst-eq*
by *fastforce*

lemma *subst-reduntant-if* [*simp*]:
assumes *vars a* ⊆ *vars'*
shows *a* · (λ*var*. *if var* ∈ *vars' then σ var else σ' var*) = *a* · *σ*
using *assms*
by (*smt* (*verit*, *best*) *subset-eq subst-eq*)

lemma *subst-reduntant-if'* [*simp*]:
assumes *vars a* ∩ *vars'* = {}
shows *a* · (λ*var*. *if var* ∈ *vars' then σ' var else σ var*) = *a* · *σ*

using *assms subst-eq*
unfolding *disjoint-iff*
by *presburger*

lemma *subst-cannot-unground*:
assumes ¬*is-ground* $(a \cdot \sigma)$
shows ¬*is-ground a*
using *assms* **by** *force*

end

locale *finite-variables* = *finite-set vars* **for** *vars* :: *'expression* ⇒ *'variable set*
begin

lemmas *finite-vars* = *finite-set finite-set'*
lemmas *fset-finite-vars* = *fset-finite-set*

abbreviation *finite-vars* ≡ *finite-set*

end

locale *all-subst-ident-iff-ground* =
fixes *is-ground* :: *'expression* ⇒ *bool* **and** *subst*
assumes
all-subst-ident-iff-ground: $\bigwedge a.$ *is-ground a* ⟷ (∀σ. *subst a σ* = *a*) **and**
exists-non-ident-subst:
$\bigwedge a\ s.$ *finite s* ⟹ ¬*is-ground a* ⟹ ∃σ. *subst a σ* ≠ *a* ∧ *subst a σ* ∉ *s*

locale *grounding* = *variable-substitution*
where *vars* = *vars* **for** *vars* :: *'a* ⇒ *'var set* +
fixes *to-ground* :: *'a* ⇒ *'g* **and** *from-ground* :: *'g* ⇒ *'a*
assumes
range-from-ground-iff-is-ground: {*f*. *is-ground f*} = *range from-ground* **and**
from-ground-inverse [*simp*]: $\bigwedge g.$ *to-ground* (*from-ground g*) = *g*
begin

definition *groundings* ::*'a* ⇒ *'g set* **where**
groundings a = { *to-ground* $(a \cdot \gamma)$ | γ. *is-ground* $(a \cdot \gamma)$ }

lemma *to-ground-from-ground-id*: *to-ground* ∘ *from-ground* = *id*
using *from-ground-inverse*
by *auto*

lemma *surj-to-ground*: *surj to-ground*
using *from-ground-inverse*
by (*metis surj-def*)

lemma *inj-from-ground*: *inj-on from-ground* $domain_G$
by (*metis from-ground-inverse inj-on-inverseI*)

lemma *inj-on-to-ground*: *inj-on to-ground* (*from-ground* ' $domain_G$)
  unfolding *inj-on-def*
  by *simp*

lemma *bij-betw-to-ground*: *bij-betw to-ground* (*from-ground* ' $domain_G$) $domain_G$
  by (*smt* (*verit*, *best*) *bij-betwI'* *from-ground-inverse image-iff*)

lemma *bij-betw-from-ground*: *bij-betw from-ground* $domain_G$ (*from-ground* ' $domain_G$)
  by (*simp add*: *bij-betw-def inj-from-ground*)

lemma *ground-is-ground* [*simp*, *intro*]: *is-ground* (*from-ground g*)
  using *range-from-ground-iff-is-ground*
  by *blast*

lemma *is-ground-iff-range-from-ground*: *is-ground f* $\longleftrightarrow$ *f* $\in$ *range from-ground*
  using *range-from-ground-iff-is-ground*
  by *auto*

lemma *to-ground-inverse* [*simp*]:
  assumes *is-ground f*
  shows *from-ground* (*to-ground f*) = *f*
  using *inj-on-to-ground from-ground-inverse is-ground-iff-range-from-ground assms*
  unfolding *inj-on-def*
  by *blast*

corollary *obtain-grounding*:
  assumes *is-ground f*
  obtains *g* **where** *from-ground g* = *f*
  using *to-ground-inverse assms* **by** *blast*

end

locale *base-variable-substitution* = *variable-substitution*
  where *subst* = *subst*
  for *subst* :: *'expression* $\Rightarrow$ (*'variable* $\Rightarrow$ *'expression*) $\Rightarrow$ *'expression* (**infixl** · *70*)
+
  assumes
    is-grounding-iff-vars-grounded:
      $\bigwedge$*exp*. *is-ground* (*exp* · γ) $\longleftrightarrow$ ($\forall$ *x* $\in$ *vars exp*. *is-ground* (γ *x*)) **and**
    ground-exists: $\exists$ *exp*. *is-ground exp*
begin

lemma *obtain-ground-subst*:
  obtains γ
  where *is-ground-subst* γ
proof−
  obtain *g* **where** *is-ground g*

using *ground-exists* **by** *blast*

then have *is-ground-subst* $(\lambda$-. $g)$
by (*simp add*: *is-grounding-iff-vars-grounded is-ground-subst-def*)

then show *?thesis*
using *that*
by *simp*
qed

lemma *ground-subst-extension*:
assumes *is-ground* $(exp \cdot \gamma)$
obtains γ'
where $exp \cdot \gamma = exp \cdot \gamma'$ **and** *is-ground-subst* γ'
proof−
obtain γ'' **where**
γ'': *is-ground-subst* γ''
using *obtain-ground-subst*
by *blast*

define γ' **where**
γ': $\gamma' = (\lambda var.\ if\ var \in vars\ exp\ then\ \gamma\ var\ else\ \gamma''\ var)$

have *is-ground-subst* γ'
using *assms* γ'' *is-grounding-iff-vars-grounded*
unfolding γ' *is-ground-subst-def*
by *simp*

moreover have $exp \cdot \gamma = exp \cdot \gamma'$
unfolding γ'
using *subst-eq* **by** *presburger*

ultimately show *?thesis*
using *that*
by *blast*
qed

lemma *ground-subst-upd* [*simp*]:
assumes *is-ground update is-ground* $(exp \cdot \gamma)$
shows *is-ground* $(exp \cdot \gamma(var := update))$
using *assms is-grounding-iff-vars-grounded* **by** *auto*

lemma *variable-grounding*:
assumes *is-ground* $(t \cdot \gamma)$ $x \in vars\ t$
shows *is-ground* $(\gamma\ x)$
using *assms is-grounding-iff-vars-grounded*
by *blast*

end

locale *based-variable-substitution* =
  base: *base-variable-substitution* **where** *subst* = *base-subst* **and** *vars* = *base-vars*
+
  variable-substitution
for *base-subst base-vars* +
assumes
  ground-subst-iff-base-ground-subst [*simp*]: *is-ground-subst* $\gamma \longleftrightarrow$ *base.is-ground-subst*
γ **and**
  is-grounding-iff-vars-grounded:
    $\bigwedge$*exp. is-ground* (*exp* · γ) $\longleftrightarrow$ ($\forall$ *x* $\in$ *vars exp. base.is-ground* (γ *x*))
begin

lemma *obtain-ground-subst*:
  obtains γ
  where *is-ground-subst* γ
  using *base.obtain-ground-subst* **by** *auto*

lemma *ground-subst-extension*:
  assumes *is-ground* (*exp* · γ)
  obtains γ'
  where *exp* · γ = *exp* · γ' **and** *is-ground-subst* γ'
  using *obtain-ground-subst assms*
  by (*metis all-subst-ident-if-ground is-ground-subst-comp-right subst-comp-subst*)

lemma *ground-subst-extension*′:
  assumes *is-ground* (*exp* · γ)
  obtains γ'
  where *exp* · γ = *exp* · γ' **and** *base.is-ground-subst* γ'
  using *ground-subst-extension assms*
  by *auto*

lemma *ground-subst-upd* [*simp*]:
  assumes *base.is-ground update is-ground* (*exp* · γ)
  shows *is-ground* (*exp* · γ(*var* := *update*))
  using *base.ground-subst-upd assms is-grounding-iff-vars-grounded* **by** *simp*

lemma *ground-exists*: $\exists$ *exp. is-ground exp*
  using *base.ground-exists*
  by (*meson is-grounding-iff-vars-grounded*)

lemma *variable-grounding*:
  assumes *is-ground* (*t* · γ) *x* $\in$ *vars t*
  shows *base.is-ground* (γ *x*)
  using *assms is-grounding-iff-vars-grounded*
  by *blast*

end

2 Liftings

locale *variable-substitution-lifting* =
sub: *variable-substitution*
where *subst* = *sub-subst* **and** *vars* = *sub-vars*
for
sub-vars :: *'sub-expression* ⇒ *'variable set* **and**
sub-subst :: *'sub-expression* ⇒ (*'variable* ⇒ *'base-expression*) ⇒ *'sub-expression*
+
fixes
map :: (*'sub-expression* ⇒ *'sub-expression*) ⇒ *'expression* ⇒ *'expression* **and**
to-set :: *'expression* ⇒ *'sub-expression set*
assumes
map-comp: $\bigwedge d\, f\, g.$ *map f* (*map g d*) = *map* ($f \circ g$) *d* **and**
map-id: *map id d* = *d* **and**
map-cong: $\bigwedge d\, f\, g.$ ($\bigwedge c.$ $c \in$ *to-set d* $\Longrightarrow$ *f c* = *g c*) $\Longrightarrow$ *map f d* = *map g d*
and
to-set-map: $\bigwedge d\, f.$ *to-set* (*map f d*) = *f* ‘ *to-set d* **and**
exists-expression: $\bigwedge c.$ $\exists\, d.$ $c \in$ *to-set d*
begin

definition *vars* :: *'expression* ⇒ *'variable set* **where**
vars d ≡ $\bigcup$ (*sub-vars* ‘ *to-set d*)

definition *subst* :: *'expression* ⇒ (*'variable* ⇒ *'base-expression*) ⇒ *'expression*
where
subst d σ ≡ *map* ($\lambda c.$ *sub-subst c* σ) *d*

lemma *map-id-cong*:
assumes $\bigwedge c.$ $c \in$ *to-set d* $\Longrightarrow$ *f c* = *c*
shows *map f d* = *d*
using *map-cong map-id assms*
unfolding *id-def*
by *metis*

lemma *to-set-map-not-ident*:
assumes $c \in$ *to-set d f c* $\notin$ *to-set d*
shows *map f d* ≠ *d*
using *assms*
by (*metis rev-image-eqI to-set-map*)

lemma *subst-in-to-set-subst*:
assumes $c \in$ *to-set d*
shows *sub-subst c* σ $\in$ *to-set* (*subst d* σ)
unfolding *subst-def*
using *assms to-set-map* **by** *auto*

sublocale *variable-substitution* **where** *subst* = *subst* **and** *vars* = *vars*
proof *unfold-locales*

```
  show ⋀x a b. subst x (comp-subst a b) = subst (subst x a) b
    using sub.subst-comp-subst
    unfolding subst-def map-comp comp-apply
    by presburger
next
  show ⋀x. subst x id-subst = x
    using map-id
    unfolding subst-def sub.subst-id-subst id-def.
next
   show ⋀x. vars x = {} ⟹ ∀σ. subst x σ = x
     unfolding vars-def subst-def
     using map-id-cong
     by simp
next
  show ⋀a σ τ. (⋀x. x ∈ vars a ⟹ σ x = τ x) ⟹ subst a σ = subst a τ
    unfolding vars-def subst-def
    using map-cong sub.subst-eq
    by (meson UN-I)
qed

lemma ground-subst-iff-sub-ground-subst [simp]:
  is-ground-subst γ ⟷ sub.is-ground-subst γ
proof(unfold is-ground-subst-def sub.is-ground-subst-def, intro iffI allI)
  fix c
  assume all-d-ground: ∀d. is-ground (subst d γ)
  show sub.is-ground (sub-subst c γ)
  proof(rule ccontr)
    assume c-not-ground: ¬sub.is-ground (sub-subst c γ)

    then obtain d where c ∈ to-set d
      using exists-expression by auto

    then have ¬is-ground (subst d γ)
      using c-not-ground to-set-map
      unfolding subst-def vars-def
      by auto

    then show False
      using all-d-ground
      by blast
  qed
next
  fix d
  assume all-c-ground: ∀c. sub.is-ground (sub-subst c γ)

  then show is-ground (subst d γ)
    unfolding vars-def subst-def
    using to-set-map
    by simp
```

qed

lemma *to-set-is-ground* [*intro*]:
assumes *sub* ∈ *to-set expr is-ground expr*
shows *sub.is-ground sub*
using *assms*
by (*simp add*: *vars-def*)

lemma *to-set-is-ground-subst*:
assumes *sub* ∈ *to-set expr is-ground* (*subst expr* γ)
shows *sub.is-ground* (*sub-subst sub* γ)
using *assms*
by (*meson subst-in-to-set-subst to-set-is-ground*)

lemma *subst-empty*:
assumes *to-set expr'* = {}
shows *subst expr* σ = *expr'* ⟷ *expr* = *expr'*
using *assms map-id-cong subst-def to-set-map*
by *fastforce*

lemma *empty-is-ground*:
assumes *to-set expr* = {}
shows *is-ground expr*
using *assms*
by (*simp add*: *vars-def*)

end

locale *based-variable-substitution-lifting* =
variable-substitution-lifting +
base: *base-variable-substitution* **where** *subst* = *base-subst* **and** *vars* = *base-vars*
for *base-subst base-vars* +
assumes
sub-is-grounding-iff-vars-grounded:
⋀*exp* γ. *sub.is-ground* (*sub-subst exp* γ) ⟷ (∀ *x* ∈ *sub-vars exp*. *base.is-ground*
(γ *x*)) **and**
sub-ground-subst-iff-base-ground-subst: ⋀γ. *sub.is-ground-subst* γ ⟷ *base.is-ground-subst*
γ
begin

lemma *is-grounding-iff-vars-grounded*:
is-ground (*subst exp* γ) ⟷ (∀ *x* ∈ *vars exp*. *base.is-ground* (γ *x*))
using *sub-is-grounding-iff-vars-grounded subst-def to-set-map vars-def*
by *auto*

lemma *ground-subst-iff-base-ground-subst* [*simp*]:
⋀γ. *is-ground-subst* γ ⟷ *base.is-ground-subst* γ
using *sub-ground-subst-iff-base-ground-subst ground-subst-iff-sub-ground-subst* **by**
blast

lemma *obtain-ground-subst*:
  obtains γ
  where *is-ground-subst* γ
  using *base.obtain-ground-subst*
  by (*meson base.ground-exists is-grounding-iff-vars-grounded is-ground-subst-def that*)

lemma *ground-subst-extension*:
  assumes *is-ground* (*subst exp* γ)
  obtains γ'
  where *subst exp* γ = *subst exp* γ' **and** *is-ground-subst* γ'
  by (*metis all-subst-ident-if-ground assms comp-subst.left.monoid-action-compatibility*
      is-ground-subst-comp-right obtain-ground-subst)

lemma *ground-subst-extension'*:
  assumes *is-ground* (*subst exp* γ)
  obtains γ'
  where *subst exp* γ = *subst exp* γ' **and** *base.is-ground-subst* γ'
  by (*metis all-subst-ident-if-ground assms base.is-ground-subst-comp-right*
      base.obtain-ground-subst subst-comp-subst)

lemma *ground-subst-upd* [*simp*]:
  assumes *base.is-ground update is-ground* (*subst exp* γ)
  shows *is-ground* (*subst exp* (γ(*var* := *update*)))
  using *assms*(*1*) *assms*(*2*) *is-grounding-iff-vars-grounded* **by** *auto*

lemma *ground-exists*: ∃ *exp. is-ground exp*
  using *base.ground-exists*
  by (*meson is-grounding-iff-vars-grounded*)

lemma *variable-grounding*:
  assumes *is-ground* (*subst t* γ) $x \in$ *vars t*
  shows *base.is-ground* (γ *x*)
  using *assms is-grounding-iff-vars-grounded*
  by *blast*

end

locale *finite-variables-lifting* =
  variable-substitution-lifting +
  sub: *finite-variables* **where** *vars* = *sub-vars* +
  to-set: *finite-set* **where** *set* = *to-set*
begin

abbreviation *to-fset* :: *'d* ⇒ *'c fset* **where**
  to-fset ≡ *to-set.finite-set*

lemmas *finite-to-set* = *to-set.finite-set to-set.finite-set'*
lemmas *fset-to-fset* = *to-set.fset-finite-set*

sublocale *finite-variables* **where** *vars* = *vars*
by *unfold-locales* (*simp add*: *vars-def*)

end

locale *grounding-lifting* =
variable-substitution-lifting **where** *sub-vars* = *sub-vars* **and** *sub-subst* = *sub-subst* **and** *map* = *map* +
sub: *grounding* **where** *vars* = *sub-vars* **and** *subst* = *sub-subst* **and** *to-ground* = *sub-to-ground* **and**
from-ground = *sub-from-ground*
for
sub-to-ground :: *'sub* ⇒ *'ground-sub* **and**
sub-from-ground :: *'ground-sub* ⇒ *'sub* **and**
sub-vars :: *'sub* ⇒ *'variable set* **and**
sub-subst :: *'sub* ⇒ (*'variable* ⇒ *'base*) ⇒ *'sub* **and**
map :: (*'sub* ⇒ *'sub*) ⇒ *'expr* ⇒ *'expr* +
fixes
to-ground-map :: (*'sub* ⇒ *'ground-sub*) ⇒ *'expr* ⇒ *'ground-expr* **and**
from-ground-map :: (*'ground-sub* ⇒ *'sub*) ⇒ *'ground-expr* ⇒ *'expr* **and**
ground-map :: (*'ground-sub* ⇒ *'ground-sub*) ⇒ *'ground-expr* ⇒ *'ground-expr* **and**
to-set-ground :: *'ground-expr* ⇒ *'ground-sub set*
assumes
to-set-from-ground-map: ⋀*d f*. *to-set* (*from-ground-map f d*) = *f* ‘ *to-set-ground d* **and**
map-comp': ⋀*d f g*. *from-ground-map f* (*to-ground-map g d*) = *map* (*f* ∘ *g*) *d* **and**
ground-map-comp: ⋀*d f g*. *to-ground-map f* (*from-ground-map g d*) = *ground-map* (*f* ∘ *g*) *d* **and**
ground-map-id: *ground-map id g* = *g*
begin

definition *to-ground* **where** *to-ground expr* ≡ *to-ground-map sub-to-ground expr*

definition *from-ground* **where** *from-ground expr* ≡ *from-ground-map sub-from-ground expr*

sublocale *grounding* **where**
vars = *vars* **and** *subst* = *subst* **and** *to-ground* = *to-ground* **and** *from-ground* = *from-ground*
proof *unfold-locales*
have ⋀*expr*. *vars expr* = {} ⟹ *expr* ∈ *range from-ground*
proof−
fix *expr*
assume *vars expr* = {}
then have ∀ *sub*∈*to-set expr*. *sub* ∈ *range sub-from-ground*

by (*simp add*: *sub.is-ground-iff-range-from-ground vars-def*)

then have ∀ *sub*∈*to-set expr*. ∃ *sub-ground*. *sub-from-ground sub-ground* = *sub*
by *fast*

then have ∃ *ground-expr*. *from-ground ground-expr* = *expr*
using *map-comp*′[*symmetric*] *map-id-cong*
unfolding *from-ground-def comp-def*
by *metis*

then show *expr* ∈ *range from-ground*
unfolding *from-ground-def*
by *blast*
qed

moreover have ⋀*expr x*. *x* ∈ *vars* (*from-ground expr*) ⟹ *False*
proof−
fix *expr x*
assume *x* ∈ *vars* (*from-ground expr*)
then show *False*
unfolding *vars-def from-ground-def*
using *sub.ground-is-ground to-set-from-ground-map* **by** *auto*
qed

ultimately show {*f*. *vars f* = {}} = *range from-ground*
by *blast*
next
show ⋀*g*. *to-ground* (*from-ground g*) = *g*
using *ground-map-id*
unfolding *to-ground-def from-ground-def ground-map-comp sub.to-ground-from-ground-id*.
qed

lemma *to-set-from-ground*: *to-set* (*from-ground expr*) = *sub-from-ground* ‘ (*to-set-ground expr*)
unfolding *from-ground-def*
by (*simp add*: *to-set-from-ground-map*)

lemma *sub-in-ground-is-ground*:
assumes *sub* ∈ *to-set* (*from-ground expr*)
shows *sub.is-ground sub*
using *assms*
by (*simp add*: *to-set-is-ground*)

lemma *ground-sub-in-ground*:
sub ∈ *to-set-ground expr* ⟷ *sub-from-ground sub* ∈ *to-set* (*from-ground expr*)
by (*simp add*: *inj-image-mem-iff sub.inj-from-ground to-set-from-ground*)

lemma *ground-sub*:
(∀ *sub* ∈ *to-set* (*from-ground* $expr_G$). *P sub*) ⟷

(∀ sub_G ∈ *to-set-ground* $expr_G$. *P* (*sub-from-ground* sub_G))
  by (*simp add*: *to-set-from-ground*)

end

locale *all-subst-ident-iff-ground-lifting* =
  finite-variables-lifting +
  sub: *all-subst-ident-iff-ground* **where** *subst* = *sub-subst* **and** *is-ground* = *sub.is-ground*
begin

sublocale *all-subst-ident-iff-ground*
  where *subst* = *subst* **and** *is-ground* = *is-ground*
proof *unfold-locales*
  show ⋀*x*. *is-ground x* = (∀ σ. *subst x* σ = *x*)
  proof(*rule iffI allI*)
    show ⋀*x*. *is-ground x* ⟹ ∀ σ. *subst x* σ = *x*
      by *simp*
  next
    fix *d x*
    assume *all-subst-ident*: ∀ σ. *subst d* σ = *d*

    show *is-ground d*
    proof(*rule ccontr*)
      assume ¬*is-ground d*

      then obtain *c* **where** *c-in-d*: *c* ∈ *to-set d* **and** *c-not-ground*: ¬*sub.is-ground*
c
        unfolding *vars-def*
        by *blast*

      then obtain σ **where** *sub-subst c* σ ≠ *c* **and** *sub-subst c* σ ∉ *to-set d*
        using *sub.exists-non-ident-subst finite-to-set*
        by *blast*

      then show *False*
        using *all-subst-ident c-in-d to-set-map*
        unfolding *subst-def*
        by (*metis image-eqI*)
    qed
  qed
next
  fix *d* :: ′*d* **and** *ds* :: ′*d set*
  assume *finite-ds*: *finite ds* **and** *d-not-ground*: ¬*is-ground d*

  then have *finite-cs*: *finite* (⋃(*to-set* ‘ *insert d ds*))
    using *finite-to-set* **by** *blast*

  obtain *c* **where** *c-in-d*: *c* ∈ *to-set d* **and** *c-not-ground*: ¬*sub.is-ground c*
    using *d-not-ground*

unfolding *vars-def*
by *blast*

obtain σ **where** σ*-not-ident*: *sub-subst c* $\sigma \neq$ *c sub-subst c* $\sigma \notin \bigcup$ (*to-set* ' *insert d ds*)
using *sub.exists-non-ident-subst*[*OF finite-cs c-not-ground*]
by *blast*

then have *subst d* $\sigma \neq d$
using *c-in-d*
unfolding *subst-def*
by (*simp add*: *to-set-map-not-ident*)

moreover have *subst d* $\sigma \notin ds$
using σ*-not-ident*(*2*) *c-in-d to-set-map*
unfolding *subst-def*
by *auto*

ultimately show $\exists\sigma$. *subst d* $\sigma \neq d \wedge$ *subst d* $\sigma \notin ds$
by *blast*
qed

end

end
theory *First-Order-Clause*
imports
Ground-Clause
Abstract-Substitution.Substitution-First-Order-Term
Variable-Substitution
Clausal-Calculus-Extra
Multiset-Extra
Term-Rewrite-System
Term-Ordering-Lifting
HOL−Eisbach.Eisbach
HOL-Extra
begin

no-notation *subst-compose* (**infixl** $\circ_s$ *75*)
no-notation *subst-apply-term* (**infixl** $\cdot$ *67*)

Prefer *term-subst.subst-id-subst* to *subst-apply-term-empty.*

declare *subst-apply-term-empty*[*no-atp*]

3 `First_Order_Terms` And `Abstract_Substitution`

type-synonym *'f ground-term* = *'f gterm*

type-synonym *'f ground-context* = *'f gctxt*

type-synonym $('f, 'v)$ *context* = $('f, 'v)$ *ctxt*

type-synonym *'f ground-atom* = *'f gatom*
type-synonym $('f, 'v)$ *atom* = $('f, 'v)$ *term uprod*

notation *subst-apply-term* (**infixl** $\cdot t$ *67*)
notation *subst-compose* (**infixl** $\odot$ *75*)

notation *subst-apply-ctxt* (**infixl** $\cdot t_c$ *67*)

lemmas *clause-simp-term* =
subst-apply-term-ctxt-apply-distrib vars-term-ctxt-apply literal.sel

named-theorems *clause-simp*
named-theorems *clause-intro*

lemma *ball-set-uprod* [*clause-simp*]: $(\forall t \in$ *set-uprod* (*Upair* t_1 t_2). $P\ t) \longleftrightarrow P\ t_1 \wedge P\ t_2$
by *auto*

lemma *infinite-terms* [*clause-intro*]: *infinite* (*UNIV* :: $('f, 'v)$ *term set*)
proof−
have *infinite* (*UNIV* :: $('f, 'v)$ *term list set*)
using *infinite-UNIV-listI*.

then have $\bigwedge f$ :: *'f*. *infinite* ((*Fun f*) ‘ (*UNIV* :: $('f, 'v)$ *term list set*))
by (*meson finite-imageD injI term.inject*(*2*))

then show *infinite* (*UNIV* :: $('f, 'v)$ *term set*)
using *infinite-super top-greatest* **by** *blast*
qed

lemma *literal-cases*: $[\![\mathcal{P} \in \{Pos, Neg\}; \mathcal{P} = Pos \Longrightarrow P; \mathcal{P} = Neg \Longrightarrow P]\!] \Longrightarrow P$
by *blast*

method *clause-simp* **uses** *simp intro* =

auto simp only: *simp clause-simp clause-simp-term intro*: *intro clause-intro*

method *clause-auto* **uses** *simp intro* =
(*clause-simp simp*: *simp intro*: *intro*)?,
(*auto simp*: *simp intro intro*)?,
(*auto simp*: *simp clause-simp intro*: *intro clause-intro*)?

locale *vars-def* =
fixes *vars-def* :: *'expression* $\Rightarrow$ *'variables*

begin

abbreviation *vars* ≡ *vars-def*

end

locale *grounding-def* =
  fixes
    to-ground-def :: *'non-ground* ⇒ *'ground* **and**
    from-ground-def :: *'ground* ⇒ *'non-ground*
begin

abbreviation *to-ground* ≡ *to-ground-def*

abbreviation *from-ground* ≡ *from-ground-def*

end

4 Term

global-interpretation *term*: *vars-def* **where** *vars-def* = *vars-term*.

global-interpretation *context*: *vars-def* **where**
  vars-def = *vars-ctxt*.

global-interpretation *term*: *grounding-def* **where**
  to-ground-def = *gterm-of-term* **and** *from-ground-def* = *term-of-gterm* .

global-interpretation *context*: *grounding-def* **where**
  to-ground-def = *gctxt-of-ctxt* **and** *from-ground-def* = *ctxt-of-gctxt*.

global-interpretation
  term: *base-variable-substitution* **where**
  subst = *subst-apply-term* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
  vars = *term.vars* :: (*'f*, *'v*) *term* ⇒ *'v set* +
  term: *finite-variables* **where** *vars* = *term.vars* :: (*'f*, *'v*) *term* ⇒ *'v set* +
  term: *all-subst-ident-iff-ground* **where**
  is-ground = *term.is-ground* :: (*'f*, *'v*) *term* ⇒ *bool* **and** *subst* = (·*t*)
proof *unfold-locales*
  show $\bigwedge t\ \sigma\ \tau.\ (\bigwedge x.\ x \in term.vars\ t \Longrightarrow \sigma\ x = \tau\ x) \Longrightarrow t \cdot t\ \sigma = t \cdot t\ \tau$
    using *term-subst-eq*.
next
  fix *t* :: (*'f*, *'v*) *term*
  show *finite* (*term.vars t*)
    by *simp*
next
  fix *t* :: (*'f*, *'v*) *term*
  show (*term.vars t* = {}) = (∀ σ. *t* ·*t* σ = *t*)
    using *is-ground-trm-iff-ident-forall-subst*.

```
next
  fix t :: ('f, 'v) term and ts :: ('f, 'v) term set

  assume finite ts term.vars t ≠ {}
  then show ∃σ. t ·t σ ≠ t ∧ t ·t σ ∉ ts
  proof(induction t arbitrary: ts)
    case (Var x)

    obtain t' where t': t' ∉ ts is-Fun t'
      using Var.prems(1) finite-list by blast

    define σ :: ('f, 'v) subst where ⋀x. σ x = t'

    have Var x ·t σ ≠ Var x
      using t'
      unfolding σ-def
      by auto

    moreover have Var x ·t σ ∉ ts
      using t'
      unfolding σ-def
      by simp

    ultimately show ?case
      using Var
      by blast
  next
    case (Fun f args)

    obtain a where a: a ∈ set args and a-vars: term.vars a ≠ {}
      using Fun.prems by fastforce

    then obtain σ where
      σ: a ·t σ ≠ a and
      a-σ-not-in-args: a ·t σ ∉ ⋃ (set ` term.args ` ts)
      by (metis Fun.IH Fun.prems(1) List.finite-set finite-UN finite-imageI)

    then have Fun f args ·t σ ≠ Fun f args
     by (metis a subsetI term.set-intros(4) term-subst.comp-subst.left.action-neutral

          vars-term-subset-subst-eq)

    moreover have Fun f args ·t σ ∉ ts
      using a a-σ-not-in-args
      by auto

    ultimately show ?case
      using Fun
      by blast
```

qed
next
show $\bigwedge\gamma\ t.\ (term.vars\ (t \cdot t\ \gamma) = \{\}) = (\forall x \in term.vars\ t.\ term.vars\ (\gamma\ x) = \{\})$
by (*meson is-ground-iff*)
next
show $\exists\, t.\ term.vars\ t = \{\}$
by (*meson vars-term-of-gterm*)
qed

lemma *term-context-ground-iff-term-is-ground* [*clause-simp*]:
Term-Context.ground t = *term.is-ground t*
by(*induction t*) *simp-all*

global-interpretation
term: *grounding* **where**
vars = *term.vars* :: ('*f*, '*v*) *term* ⇒ '*v set* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
subst = (·*t*) **and** *to-ground* = *term.to-ground* **and** *from-ground* = *term.from-ground*
proof *unfold-locales*
have $\bigwedge t$ :: ('*f*, '*v*) *term*. *term.is-ground t* ⟹ $\exists\, g$. *term.from-ground g* = *t*
proof(*intro exI*)
fix *t* :: ('*f*, '*v*) *term*
assume *term.is-ground t*
then show *term.from-ground* (*term.to-ground t*) = *t*
by(*induction t*)(*simp-all add*: *map-idI*)
qed

then show {*t* :: ('*f*, '*v*) *term*. *term.is-ground t*} = *range term.from-ground*
by *fastforce*
next
show $\bigwedge g$. *term.to-ground* (*term.from-ground g*) = *g*
by *simp*
qed

global-interpretation *context*: *all-subst-ident-iff-ground* **where**
is-ground = $\lambda\kappa$. *context.vars* $\kappa = \{\}$ **and** *subst* = $(\cdot t_c)$
proof *unfold-locales*
fix κ :: ('*f*, '*v*) *context*
show *context.vars* $\kappa = \{\} = (\forall\sigma.\ \kappa \cdot t_c\ \sigma = \kappa)$
proof (*intro iffI*)
show *context.vars* $\kappa = \{\} \Longrightarrow \forall\sigma.\ \kappa \cdot t_c\ \sigma = \kappa$
by(*induction* κ) (*simp-all add*: *list.map-ident-strong*)
next
assume $\forall\sigma.\ \kappa \cdot t_c\ \sigma = \kappa$

then have $\bigwedge t_G$. *term.is-ground* $t_G \Longrightarrow \forall\sigma.\ \kappa\langle t_G\rangle \cdot t\ \sigma = \kappa\langle t_G\rangle$
by *simp*

then have $\bigwedge t_G$. *term.is-ground* $t_G \Longrightarrow$ *term.is-ground* $\kappa\langle t_G\rangle$
  by (*meson is-ground-trm-iff-ident-forall-subst*)

then show *context.vars* $\kappa = \{\}$
  by (*metis sup.commute sup-bot-left vars-term-ctxt-apply vars-term-of-gterm*)
qed
next
fix κ :: ($'f$, $'v$) *context* **and** κs :: ($'f$, $'v$) *context set*
assume *finite*: *finite* κs **and** *non-ground*: *context.vars* $\kappa \neq \{\}$

then show $\exists\sigma.\ \kappa \cdot t_c\ \sigma \neq \kappa \land \kappa \cdot t_c\ \sigma \notin \kappa s$
proof(*induction* κ *arbitrary*: κs)
  case *Hole*
  then show *?case*
    by *simp*
next
  case (*More f ts* κ *ts'*)

  show *?case*
  proof(*cases context.vars* $\kappa = \{\}$)
    case *True*

    let *?sub-terms* =
      $\lambda\kappa$ :: ($'f$, $'v$) *context. case* κ *of More - ts - ts'* $\Rightarrow$ *set ts* $\cup$ *set ts'* | - $\Rightarrow$ {}

    let *?*$\kappa s'$ = *set ts* $\cup$ *set ts'* $\cup \bigcup$(*?sub-terms* ‘ κs)

    from *True* **obtain** *t* **where** *t*: $t \in$ *set ts* $\cup$ *set ts'* **and** *non-ground*: ¬*term.is-ground*
t
      using *More.prems* **by** *auto*

    have $\bigwedge\kappa$. *finite* (*?sub-terms* κ)
    proof−
      fix κ
      show *finite* (*?sub-terms* κ)
        by(*cases* κ) *simp-all*
    qed

    then have *finite* ($\bigcup$(*?sub-terms* ‘ κs))
      using *More.prems*(*1*) **by** *blast*

    then have *finite*: *finite ?*$\kappa s'$
      by *blast*

    obtain σ **where** σ: $t \cdot t\ \sigma \neq t$ **and** $\kappa s'$: $t \cdot t\ \sigma \notin$ *?*$\kappa s'$
      using *term.exists-non-ident-subst*[*OF finite non-ground*]
      by *blast*

    then have *More f ts* κ *ts'* $\cdot t_c\ \sigma \neq$ *More f ts* κ *ts'*

using t *set-map-id*[*of* - - $\lambda t.\ t \cdot t\ \sigma$]
by *auto*

moreover have *More f ts* κ $ts' \cdot t_c\ \sigma \notin \kappa s$
using $\kappa s'$ t
by *auto*

ultimately show *?thesis*
by *blast*
next
case *False*

let *?sub-contexts* $= (\lambda\kappa.\ case\ \kappa\ of\ More$ - - κ - $\Rightarrow \kappa)$ ‘ $\{\kappa \in \kappa s.\ \kappa \neq \square\}$

have *finite ?sub-contexts*
using *More.prems*(*1*)
by *auto*

then obtain σ **where** σ: $\kappa \cdot t_c\ \sigma \neq \kappa$ **and** *sub-contexts*: $\kappa \cdot t_c\ \sigma \notin$ *?sub-contexts*
using *More.IH*[*OF* - *False*]
by *blast*

then have *More f ts* κ $ts' \cdot t_c\ \sigma \neq$ *More f ts* κ ts'
by *simp*

moreover have *More f ts* κ $ts' \cdot t_c\ \sigma \notin \kappa s$
using *sub-contexts image-iff*
by *fastforce*

ultimately show *?thesis*
by *blast*
qed
qed
qed

global-interpretation *context*: *based-variable-substitution* **where**
subst $= (\cdot t_c)$ **and** *vars* = *context.vars* **and** *id-subst* = *Var* **and** *comp-subst* = $(\odot)$ **and**
base-vars = *term.vars* **and** *base-subst* $= (\cdot t)$
proof(*unfold-locales*, *unfold substitution-ops.is-ground-subst-def*)
fix κ :: $('f,\ 'v)$ *context*
show $\kappa \cdot t_c$ *Var* $= \kappa$
by (*induction* κ) *auto*
next
show $\bigwedge\kappa\ \sigma\ \tau.\ \kappa \cdot t_c\ \sigma \odot \tau = \kappa \cdot t_c\ \sigma \cdot t_c\ \tau$
by *simp*
next
show $\bigwedge\kappa.$ *context.vars* $\kappa = \{\} \Longrightarrow \forall\sigma.\ \kappa \cdot t_c\ \sigma = \kappa$
using *context.all-subst-ident-iff-ground* **by** *blast*

next
  show $\bigwedge a\ \sigma\ \tau.\ (\bigwedge x.\ x \in \mathit{context.vars}\ a \Longrightarrow \sigma\ x = \tau\ x) \Longrightarrow a \cdot t_c\ \sigma = a \cdot t_c\ \tau$
    using *ctxt-subst-eq*.
next
  fix γ :: $('f,'v)$ *subst*

  show $(\forall x.\ \mathit{context.vars}\ (x \cdot t_c\ \gamma) = \{\}) \longleftrightarrow (\forall x.\ \mathit{term.vars}\ (x \cdot t\ \gamma) = \{\})$
  proof(*intro iffI allI equals0I*)
    fix *t x*

    assume *is-ground*: $\forall \kappa.\ \mathit{context.vars}\ (\kappa \cdot t_c\ \gamma) = \{\}$ **and** *vars*: $x \in \mathit{term.vars}\ (t \cdot t\ \gamma)$

    have $\bigwedge f.\ \mathit{context.vars}\ (\mathit{More}\ f\ [t]\ \mathit{Hole}\ []\ \cdot t_c\ \gamma) = \{\}$
      using *is-ground*
      by *presburger*

    moreover have $\bigwedge f.\ x \in \mathit{context.vars}\ (\mathit{More}\ f\ [t]\ \mathit{Hole}\ []\ \cdot t_c\ \gamma)$
      using *vars*
      by *simp*

    ultimately show *False*
      by *blast*
  next
    fix κ *x*
    assume *is-ground*: $\forall t.\ \mathit{term.is\text{-}ground}\ (t \cdot t\ \gamma)$ **and** *vars*: $x \in \mathit{context.vars}\ (\kappa \cdot t_c\ \gamma)$

    have $\bigwedge t.\ \mathit{term.is\text{-}ground}\ (\kappa\langle t\rangle \cdot t\ \gamma)$
      using *is-ground*
      by *presburger*

    moreover have $\bigwedge t.\ x \in \mathit{term.vars}\ (\kappa\langle t\rangle \cdot t\ \gamma)$
      using *vars*
      by *simp*

    ultimately show *False*
      by *blast*
  qed
next
  fix κ **and** γ :: $('f, 'v)$ *subst*

  show $\mathit{context.vars}\ (\kappa \cdot t_c\ \gamma) = \{\} \longleftrightarrow (\forall x \in \mathit{context.vars}\ \kappa.\ \mathit{term.is\text{-}ground}\ (\gamma\ x))$
    by(*induction* κ)(*auto simp*: *term.is-grounding-iff-vars-grounded*)
qed

global-interpretation *context*: *finite-variables*
  where $\mathit{vars} = \mathit{context.vars} :: ('f, 'v)\ \mathit{context} \Rightarrow 'v\ \mathit{set}$

proof *unfold-locales*
fix κ :: ('*f*, '*v*) *context*

have $\bigwedge t$. *finite* (*term.vars* $\kappa\langle t\rangle$)
using *term.finite-vars* **by** *blast*

then show *finite* (*context.vars* κ)
unfolding *vars-term-ctxt-apply finite-Un*
by *simp*
qed

global-interpretation *context*: *grounding* **where**
vars = *context.vars* :: ('*f*, '*v*) *context* ⇒ '*v set* **and** *id-subst* = *Var* **and** *comp-subst* = ($\odot$) **and**
subst = ($\cdot t_c$) **and** *from-ground* = *context.from-ground* **and** *to-ground* = *context.to-ground*
proof *unfold-locales*
have $\bigwedge x$. *context.vars* x = {} $\Longrightarrow \exists\, g$. *context.from-ground* g = x
by (*metis Un-empty-left gctxt-of-ctxt-inv term.ground-exists term.to-ground-inverse*

term-of-gterm-ctxt-apply-ground(*1*) *vars-term-ctxt-apply*)

then show {f. *context.vars* f = {}} = *range context.from-ground*
by *force*
next
show $\bigwedge g$. *context.to-ground* (*context.from-ground* g) = g
by *simp*
qed

lemma *ground-ctxt-iff-context-is-ground* [*clause-simp*]:
ground-ctxt context ⟷ *context.is-ground context*
by(*induction context*) *clause-auto*

5 Lifting

lemma *exists-uprod*: $\exists\, a$. $t \in$ *set-uprod a*
by (*metis insertI1 set-uprod-simps*)

lemma *exists-literal*: $\exists\, l$. $a \in$ *set-literal l*
by (*meson literal.set-intros*(*1*))

lemma *exists-mset*: $\exists\, c$. $l \in$ *set-mset c*
by (*meson union-single-eq-member*)

lemma *finite-set-literal*: $\bigwedge l$. *finite* (*set-literal l*)
unfolding *set-literal-atm-of*
by *simp*

locale *clause-lifting* =

based-variable-substitution-lifting **where**
base-subst = (·*t*) **and** *base-vars* = *term.vars* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) +
all-subst-ident-iff-ground-lifting **where** *id-subst* = *Var* **and** *comp-subst* = (⊙) +
grounding-lifting **where** *id-subst* = *Var* **and** *comp-subst* = (⊙)

global-interpretation *atom*: *clause-lifting* **where**
sub-subst = (·*t*) **and** *sub-vars* = *term.vars* **and** *map* = *map-uprod* **and** *to-set* = *set-uprod* **and**
sub-to-ground = *term.to-ground* **and** *sub-from-ground* = *term.from-ground* **and**
to-ground-map = *map-uprod* **and** *from-ground-map* = *map-uprod* **and** *ground-map* = *map-uprod* **and**
to-set-ground = *set-uprod*
by
unfold-locales
(*auto*
simp: *uprod.map-comp uprod.map-id uprod.set-map exists-uprod*
term.is-grounding-iff-vars-grounded
intro: *uprod.map-cong*)

global-interpretation *literal*: *clause-lifting* **where**
sub-subst = *atom.subst* **and** *sub-vars* = *atom.vars* **and** *map* = *map-literal* **and**
to-set = *set-literal* **and** *sub-to-ground* = *atom.to-ground* **and**
sub-from-ground = *atom.from-ground* **and** *to-ground-map* = *map-literal* **and**
from-ground-map = *map-literal* **and** *ground-map* = *map-literal* **and** *to-set-ground* = *set-literal*
by
unfold-locales
(*auto*
simp: *literal.map-comp literal.map-id literal.set-map exists-literal*
atom.is-grounding-iff-vars-grounded finite-set-literal
intro: *literal.map-cong*)

global-interpretation *clause*: *clause-lifting* **where**
sub-subst = *literal.subst* **and** *sub-vars* = *literal.vars* **and** *map* = *image-mset* **and**

to-set = *set-mset* **and** *sub-to-ground* = *literal.to-ground* **and**
sub-from-ground = *literal.from-ground* **and** *to-ground-map* = *image-mset* **and**
from-ground-map = *image-mset* **and** *ground-map* = *image-mset* **and** *to-set-ground* = *set-mset*
by *unfold-locales*
(*auto simp*: *exists-mset literal.is-grounding-iff-vars-grounded*)

notation *atom.subst* (**infixl** ·*a 67*)
notation *literal.subst* (**infixl** ·*l 66*)
notation *clause.subst* (**infixl** · *67*)

lemmas [*clause-simp*] = *literal.to-set-is-ground atom.to-set-is-ground*
lemmas [*clause-intro*] = *clause.subst-in-to-set-subst*

lemmas *empty-clause-is-ground* [*clause-intro*] =
  clause.empty-is-ground[*OF set-mset-empty*]

lemmas *clause-subst-empty* [*clause-simp*] =
  clause.subst-ident-if-ground[*OF empty-clause-is-ground*]
  clause.subst-empty[*OF set-mset-empty*]

lemma *set-mset-set-uprod* [*clause-simp*]: *set-mset* (*mset-lit literal*) = *set-uprod* (*atm-of literal*)
  by(*cases literal*) *simp-all*

lemma *mset-lit-set-literal* [*clause-simp*]:
  $term \in\# \ mset\text{-}lit\ literal \longleftrightarrow term \in \bigcup(set\text{-}uprod\ `\ set\text{-}literal\ literal)$
  unfolding *set-literal-atm-of*
  by *clause-simp*

lemma *vars-atom* [*clause-simp*]:
  $atom.vars\ (Upair\ term_1\ term_2) = term.vars\ term_1 \cup term.vars\ term_2$
  by (*simp-all add*: *atom.vars-def*)

lemma *vars-literal* [*clause-simp*]:
  literal.vars (*Pos atom*) = *atom.vars atom*
  literal.vars (*Neg atom*) = *atom.vars atom*
  literal.vars ((*if b then Pos else Neg*) *atom*) = *atom.vars atom*
  by (*simp-all add*: *literal.vars-def*)

lemma *subst-atom* [*clause-simp*]:
  $Upair\ term_1\ term_2 \cdot a\ \sigma = Upair\ (term_1 \cdot t\ \sigma)\ (term_2 \cdot t\ \sigma)$
  unfolding *atom.subst-def*
  by *simp-all*

lemma *subst-literal* [*clause-simp*]:
  $Pos\ atom \cdot l\ \sigma = Pos\ (atom \cdot a\ \sigma)$
  $Neg\ atom \cdot l\ \sigma = Neg\ (atom \cdot a\ \sigma)$
  $atm\text{-}of\ (literal \cdot l\ \sigma) = atm\text{-}of\ literal \cdot a\ \sigma$
  unfolding *literal.subst-def*
  using *literal.map-sel*
  by *auto*

lemma *vars-clause-add-mset* [*clause-simp*]:
  clause.vars (*add-mset literal clause*) = *literal.vars literal* ∪ *clause.vars clause*
  by (*simp add*: *clause.vars-def*)

lemma *vars-clause-plus* [*clause-simp*]:
  $clause.vars\ (clause_1 + clause_2) = clause.vars\ clause_1 \cup clause.vars\ clause_2$

by (*simp add*: *clause.vars-def*)

lemma *clause-submset-vars-clause-subset* [*clause-intro*]:
clause$_1$ ⊆# *clause*$_2$ ⟹ *clause.vars clause*$_1$ ⊆ *clause.vars clause*$_2$
by (*metis subset-mset.add-diff-inverse sup-ge1 vars-clause-plus*)

lemma *subst-clause-add-mset* [*clause-simp*]:
add-mset literal clause · σ = *add-mset* (*literal* ·*l* σ) (*clause* · σ)
unfolding *clause.subst-def*
by *simp*

lemma *subst-clause-plus* [*clause-simp*]:
(*clause*$_1$ + *clause*$_2$) · σ = *clause*$_1$ · σ + *clause*$_2$ · σ
unfolding *clause.subst-def*
by *simp*

lemma *clause-to-ground-plus* [*simp*]:
clause.to-ground (*clause*$_1$ + *clause*$_2$) = *clause.to-ground clause*$_1$ + *clause.to-ground clause*$_2$
by (*simp add*: *clause.to-ground-def*)

lemma *clause-from-ground-plus* [*simp*]:
clause.from-ground (*clause*$_{G1}$ + *clause*$_{G2}$) = *clause.from-ground clause*$_{G1}$ + *clause.from-ground clause*$_{G2}$
by (*simp add*: *clause.from-ground-def*)

lemma *subst-clause-remove1-mset* [*clause-simp*]:
assumes *literal* ∈# *clause*
shows *remove1-mset literal clause* · σ = *remove1-mset* (*literal* ·*l* σ) (*clause* · σ)
unfolding *clause.subst-def image-mset-remove1-mset-if*
using *assms*
by *simp*

lemma *sub-ground-clause* [*clause-intro*]:
assumes *clause*′ ⊆# *clause clause.is-ground clause*
shows *clause.is-ground clause*′
using *assms*
unfolding *clause.vars-def*
by *blast*

lemma *clause-from-ground-empty-mset* [*clause-simp*]: *clause.from-ground* {#} = {#}
by (*simp add*: *clause.from-ground-def*)

lemma *clause-to-ground-empty-mset* [*clause-simp*]: *clause.to-ground* {#} = {#}
by (*simp add*: *clause.to-ground-def*)

lemma *ground-term-with-context1*:
assumes *context.is-ground context term.is-ground term*

shows (*context.to-ground context*)⟨*term.to-ground term*⟩$_G$ = *term.to-ground context*⟨*term*⟩
using *assms*
by (*simp add*: *term-context-ground-iff-term-is-ground*)

lemma *ground-term-with-context2*:
assumes *context.is-ground context*
shows *term.from-ground* (*context.to-ground context*)⟨*term*$_G$⟩$_G$ = *context*⟨*term.from-ground term*$_G$⟩
using *assms*
by (*simp add*: *ground-ctxt-iff-context-is-ground ground-gctxt-of-ctxt-apply-gterm*)

lemma *ground-term-with-context3*:
(*context.from-ground context*$_G$)⟨*term.from-ground term*$_G$⟩ = *term.from-ground context*$_G$⟨*term*$_G$⟩$_G$
using *ground-term-with-context2*[*OF context.ground-is-ground*, *symmetric*]
unfolding *context.from-ground-inverse*.

lemmas *ground-term-with-context* =
ground-term-with-context1
ground-term-with-context2
ground-term-with-context3

lemma *context-is-ground-context-compose1*:
assumes *context.is-ground* (*context* $\circ_c$ *context'*)
shows *context.is-ground context context.is-ground context'*
using *assms*
by(*induction context*) *auto*

lemma *context-is-ground-context-compose2*:
assumes *context.is-ground context context.is-ground context'*
shows *context.is-ground* (*context* $\circ_c$ *context'*)
using *assms*
by (*meson ground-ctxt-comp ground-ctxt-iff-context-is-ground*)

lemmas *context-is-ground-context-compose* =
context-is-ground-context-compose1
context-is-ground-context-compose2

lemma *ground-context-subst*:
assumes
context.is-ground context$_G$
context$_G$ = (*context* $\cdot t_c$ σ) $\circ_c$ *context'*
shows
context$_G$ = *context* $\circ_c$ *context'* $\cdot t_c$ σ
using *assms*
proof(*induction context*)
case *Hole*
then show *?case*

by *simp*
next
case *More*
then show *?case*
using *context-is-ground-context-compose1*(*2*)
by (*metis subst-compose-ctxt-compose-distrib context.subst-ident-if-ground*)
qed

lemma *clause-from-ground-add-mset* [*clause-simp*]:
clause.from-ground (*add-mset* $literal_G$ $clause_G$) =
add-mset (*literal.from-ground* $literal_G$) (*clause.from-ground* $clause_G$)
by (*simp add*: *clause.from-ground-def*)

lemma *remove1-mset-literal-from-ground*:
remove1-mset (*literal.from-ground* $literal_G$) (*clause.from-ground* $clause_G$)
= *clause.from-ground* (*remove1-mset* $literal_G$ $clause_G$)
unfolding *clause.from-ground-def image-mset-remove1-mset*[*OF literal.inj-from-ground*]**..**

lemma *term-with-context-is-ground* [*clause-simp*]:
term.is-ground *context*⟨*term*⟩ ⟷ *context.is-ground* *context* ∧ *term.is-ground*
term
by *simp*

lemma *mset-literal-from-ground*:
mset-lit (*literal.from-ground* *l*) = *image-mset* *term.from-ground* (*mset-lit* *l*)
by (*metis atom.from-ground-def literal.from-ground-def literal.map-cong0 mset-lit-image-mset*)

lemma *clause-is-ground-add-mset* [*clause-simp*]:
clause.is-ground (*add-mset literal clause*) ⟷
literal.is-ground literal ∧ *clause.is-ground clause*
by *clause-auto*

lemma *clause-to-ground-add-mset*:
assumes *clause.from-ground clause* = *add-mset literal clause′*
shows *clause* = *add-mset* (*literal.to-ground literal*) (*clause.to-ground clause′*)
using *assms*
by (*metis clause.from-ground-inverse clause.to-ground-def image-mset-add-mset*)

lemma *mset-mset-lit-subst* [*clause-simp*]:
{# *term* ·*t* σ. *term* ∈# *mset-lit literal* #} = *mset-lit* (*literal* ·*l* σ)
unfolding *literal.subst-def atom.subst-def*
by (*cases literal*) (*auto simp*: *mset-uprod-image-mset*)

lemma *term-in-literal-subst* [*clause-intro*]:
assumes *term* ∈# *mset-lit literal*
shows *term* ·*t* σ ∈# *mset-lit* (*literal* ·*l* σ)
using *assms*

by (*simp add*: *atom.subst-in-to-set-subst set-mset-set-uprod subst-literal*(*3*))

lemma *ground-term-in-ground-literal*:
assumes *literal.is-ground literal term* ∈# *mset-lit literal*
shows *term.is-ground term*
by (*metis assms*(*1*,*2*) *atom.to-set-is-ground literal.simps*(*15*) *literal.vars-def set-literal-atm-of set-mset-set-uprod vars-literal*(*1*))

lemma *ground-term-in-ground-literal-subst*:
assumes *literal.is-ground* (*literal* ·*l* γ) *term* ∈# *mset-lit literal*
shows *term.is-ground* (*term* ·*t* γ)
using *assms*(*1*,*2*) *ground-term-in-ground-literal term-in-literal-subst* **by** *blast*

lemma *subst-polarity-stable*:
shows
subst-neg-stable: *is-neg* (*literal* ·*l* σ) ⟷ *is-neg literal* **and**
subst-pos-stable: *is-pos* (*literal* ·*l* σ) ⟷ *is-pos literal*
by (*simp-all add*: *literal.subst-def*)

lemma *atom-from-ground-term-from-ground* [*clause-simp*]:
atom.from-ground (*Upair* $term_{G1}$ $term_{G2}$) =
Upair (*term.from-ground* $term_{G1}$) (*term.from-ground* $term_{G2}$)
by (*simp add*: *atom.from-ground-def*)

lemma *literal-from-ground-atom-from-ground* [*clause-simp*]:
literal.from-ground (*Neg* $atom_G$) = *Neg* (*atom.from-ground* $atom_G$)
literal.from-ground (*Pos* $atom_G$) = *Pos* (*atom.from-ground* $atom_G$)
by (*simp-all add*: *literal.from-ground-def*)

lemma *context-from-ground-hole* [*clause-simp*]:
context.from-ground $context_G$ = □ ⟷ $context_G$ = $□_G$
by(*cases* $context_G$) *simp-all*

lemma *literal-from-ground-polarity-stable*:
shows
literal-from-ground-neg-stable: *is-neg* $literal_G$ ⟷ *is-neg* (*literal.from-ground* $literal_G$) **and**
literal-from-ground-stable: *is-pos* $literal_G$ ⟷ *is-pos* (*literal.from-ground* $literal_G$)
by (*simp-all add*: *literal.from-ground-def*)

lemma *ground-terms-in-ground-atom1*:
assumes *term.is-ground* $term_1$ **and** *term.is-ground* $term_2$
shows *Upair* (*term.to-ground* $term_1$) (*term.to-ground* $term_2$) = *atom.to-ground* (*Upair* $term_1$ $term_2$)
using *assms*

by (*simp add*: *atom.to-ground-def*)

lemma *ground-terms-in-ground-atom2* [*clause-simp*]:
atom.is-ground (*Upair* $term_1$ $term_2$) $\longleftrightarrow$ *term.is-ground* $term_1$ $\wedge$ *term.is-ground* $term_2$
by *clause-simp*

lemmas *ground-terms-in-ground-atom* =
ground-terms-in-ground-atom1
ground-terms-in-ground-atom2

lemma *ground-atom-in-ground-literal*:
Pos (*atom.to-ground atom*) = *literal.to-ground* (*Pos atom*)
Neg (*atom.to-ground atom*) = *literal.to-ground* (*Neg atom*)
by (*simp-all add*: *literal.to-ground-def*)

lemma *atom-is-ground-in-ground-literal* [*intro*]:
literal.is-ground literal $\longleftrightarrow$ *atom.is-ground* (*atm-of literal*)
by (*simp add*: *literal.vars-def set-literal-atm-of*)

lemma *obtain-from-atom-subst* [*clause-intro*]:
assumes *Upair* $term_1'$ $term_2'$ = *atom* $\cdot a$ σ
obtains $term_1$ $term_2$
where *atom* = *Upair* $term_1$ $term_2$ $term_1'$ = $term_1$ $\cdot t$ σ $term_2'$ = $term_2$ $\cdot t$ σ
using *assms*
unfolding *atom.subst-def*
by(*cases atom*) *auto*

lemma *obtain-from-pos-literal-subst* [*clause-intro*]:
assumes *literal* $\cdot l$ σ = $term_1'$ $\approx$ $term_2'$
obtains $term_1$ $term_2$
where *literal* = $term_1$ $\approx$ $term_2$ $term_1'$ = $term_1$ $\cdot t$ σ $term_2'$ = $term_2$ $\cdot t$ σ
using *assms obtain-from-atom-subst subst-pos-stable*
by (*metis is-pos-def literal.sel*(*1*) *subst-literal*(*1*))

lemma *obtain-from-neg-literal-subst*:
assumes *literal* $\cdot l$ σ = $term_1'$!$\approx$ $term_2'$
obtains $term_1$ $term_2$
where *literal* = $term_1$!$\approx$ $term_2$ $term_1$ $\cdot t$ σ = $term_1'$ $term_2$ $\cdot t$ σ = $term_2'$
using *assms obtain-from-atom-subst subst-neg-stable*
by (*metis literal.collapse*(*2*) *literal.disc*(*2*) *literal.sel*(*2*) *subst-literal*(*3*))

lemmas *obtain-from-literal-subst* = *obtain-from-pos-literal-subst obtain-from-neg-literal-subst*

lemma *subst-cannot-add-var*:
assumes *is-Var* (*term* $\cdot t$ σ)
shows *is-Var term*
using *assms term.subst-cannot-unground*
by *fastforce*

lemma *var-in-term*:
  assumes *var* ∈ *term.vars term*
  obtains *context* **where** *term* = *context*⟨*Var var*⟩
  using *assms*
proof(*induction term*)
  case *Var*
  then show *?case*
    by (*meson supteq-Var supteq-ctxtE*)
next
  case (*Fun f args*)
  then obtain *term′* **where** *term′* ∈ *set args* *var* ∈ *term.vars term′*
    by (*metis term.distinct*(*1*) *term.sel*(*4*) *term.set-cases*(*2*))

  moreover then obtain *args1 args2* **where**
    args1 @ [*term′*] @ *args2* = *args*
    by (*metis append-Cons append-Nil split-list*)

  moreover then have (*More f args1* □ *args2*)⟨*term′*⟩ = *Fun f args*
    by *simp*

  ultimately show *?case*
    using *Fun*(*1*)[*of term′*]
    by (*meson assms supteq-ctxtE that vars-term-supteq*)
qed

lemma *var-in-non-ground-term*:
  assumes ¬ *term.is-ground term*
  obtains *context var* **where** *term* = *context*⟨*var*⟩ *is-Var var*
proof−
  obtain *var* **where** *var* ∈ *term.vars term*
    using *assms*
    by *blast*

  moreover then obtain *context* **where** *term* = *context*⟨*Var var*⟩
    using *var-in-term*
    by *metis*

  ultimately show *?thesis*
    using *that*
    by *blast*
qed

lemma *non-ground-arg*:
  assumes ¬ *term.is-ground* (*Fun f terms*)
  obtains *term*
  where *term* ∈ *set terms* ¬ *term.is-ground term*
  using *assms that* **by** *fastforce*

lemma *non-ground-arg'*:
  assumes ¬ *term.is-ground* (*Fun f terms*)
  obtains *ts1 var ts2*
  where *terms* = *ts1* @ [*var*] @ *ts2* ¬ *term.is-ground var*
  using *non-ground-arg*
  by (*metis append.left-neutral append-Cons assms split-list*)

5.1 Interpretations

lemma *vars-term-ms-count*:
  assumes *term.is-ground* $term_G$
  shows *size* {#*var'* ∈# *vars-term-ms context*⟨*Var var*⟩. *var'* = *var*#} =
    Suc (*size* {#*var'* ∈# *vars-term-ms context*⟨$term_G$⟩. *var'* = *var*#})
proof(*induction context*)
  case *Hole*
  then show *?case*
    using *assms*
    by (*simp add*: *filter-mset-empty-conv*)
next
  case (*More f ts1 context ts2*)
  then show *?case*
    by *auto*
qed

context
  fixes *I* :: (*'f gterm* × *'f gterm*) *set*
  assumes
    trans: *trans I* **and**
    sym: *sym I* **and**
    compatible-with-gctxt: *compatible-with-gctxt I*
begin

lemma *interpretation-context-congruence*:
  assumes
    (*t*, *t'*) ∈ *I*
    (*ctxt*⟨*t*⟩$_G$, *t''*) ∈ *I*
  shows
    (*ctxt*⟨*t'*⟩$_G$, *t''*) ∈ *I*
  using
    assms sym trans compatible-with-gctxt
    compatible-with-gctxtD symE transE
  by *meson*

lemma *interpretation-context-congruence'*:
  assumes
    (*t*, *t'*) ∈ *I*
    (*ctxt*⟨*t*⟩$_G$, *t''*) ∉ *I*
  shows
    (*ctxt*⟨*t'*⟩$_G$, *t''*) ∉ *I*

using *assms sym trans compatible-with-gctxt*
by (*metis interpretation-context-congruence symD*)

context
fixes
γ :: ('*f*, '*v*) *subst* **and**
update :: ('*f*, '*v*) *Term.term* **and**
var :: '*v*
assumes
update-is-ground: *term.is-ground update* **and**
var-grounding: *term.is-ground* (*Var var* $\cdot t$ γ)
begin

lemma *interpretation-term-congruence*:
assumes
term-grounding: *term.is-ground* (*term* $\cdot t$ γ) **and**
var-update: (*term.to-ground* (γ *var*), *term.to-ground update*) $\in I$ **and**
updated-term: (*term.to-ground* (*term* $\cdot t$ γ(*var* := *update*)), *term'*) $\in I$
shows
(*term.to-ground* (*term* $\cdot t$ γ), *term'*) $\in I$
using *assms*
proof(*induction size* (*filter-mset* (λ*var'*. *var'* = *var*) (*vars-term-ms term*)) *arbitrary*: *term*)
case *0*

then have *var* $\notin$ *term.vars term*
by (*metis* (*mono-tags*, *lifting*) *filter-mset-empty-conv set-mset-vars-term-ms size-eq-0-iff-empty*)

then have *term* $\cdot t$ γ(*var* := *update*) = *term* $\cdot t$ γ
using *term.subst-reduntant-upd*
by *fast*

with *0* **show** *?case*
by *argo*
next
case (*Suc n*)

then have *var* $\in$ *term.vars term*
by (*metis* (*full-types*) *filter-mset-empty-conv nonempty-has-size set-mset-vars-term-ms*

zero-less-Suc)

then obtain *context* **where**
term [*simp*]: *term* = *context*⟨*Var var*⟩
by (*meson var-in-term*)

have [*simp*]: (*context.to-ground* (*context* $\cdot t_c$ γ))⟨*term.to-ground* (γ *var*)⟩$_G$ =
term.to-ground (*context*⟨*Var var*⟩ $\cdot t$ γ)

using *Suc* **by** *fastforce*

have *context-update* [*simp*]:
$(context.to\text{-}ground\ (context \cdot t_c\ \gamma))\langle term.to\text{-}ground\ update\rangle_G =$
$term.to\text{-}ground\ (context\langle update\rangle \cdot t\ \gamma)$
using *Suc update-is-ground*
unfolding *term*
by *auto*

have $n = size\ \{\#var' \in\# vars\text{-}term\text{-}ms\ context\langle update\rangle.\ var' = var\#\}$
using *Suc vars-term-ms-count*[*OF update-is-ground*, *of var context*]
by *auto*

moreover have $term.is\text{-}ground\ (context\langle update\rangle \cdot t\ \gamma)$
using *Suc.prems update-is-ground*
by *auto*

moreover have $(term.to\text{-}ground\ (context\langle update\rangle \cdot t\ \gamma(var := update)),\ term')$
$\in I$
using *Suc.prems update-is-ground*
by *auto*

moreover have *update*: $(term.to\text{-}ground\ update,\ term.to\text{-}ground\ (\gamma\ var)) \in I$
using *var-update sym*
by (*metis symD*)

moreover have $(term.to\text{-}ground\ (context\langle update\rangle \cdot t\ \gamma),\ term') \in I$
using *Suc calculation*
by *blast*

ultimately have $((context.to\text{-}ground\ (context \cdot t_c\ \gamma))\langle term.to\text{-}ground\ (\gamma\ var)\rangle_G,$
$term') \in I$
using *interpretation-context-congruence context-update*
by *presburger*

then show *?case*
unfolding *term*
by *simp*
qed

lemma *interpretation-term-congruence'*:
assumes
term-grounding: $term.is\text{-}ground\ (term \cdot t\ \gamma)$ **and**
var-update: $(term.to\text{-}ground\ (\gamma\ var),\ term.to\text{-}ground\ update) \in I$ **and**
updated-term: $(term.to\text{-}ground\ (term \cdot t\ \gamma(var := update)),\ term') \notin I$
shows
$(term.to\text{-}ground\ (term \cdot t\ \gamma),\ term') \notin I$
proof
assume $(term.to\text{-}ground\ (term \cdot t\ \gamma),\ term') \in I$

```
  then show False
    using
      First-Order-Clause.interpretation-term-congruence[OF
        trans sym compatible-with-gctxt var-grounding
        ]
      assms
      sym
      update-is-ground
   by (smt (verit) eval-term.simps fun-upd-same fun-upd-triv fun-upd-upd term.ground-subst-upd

          symD)
qed

lemma interpretation-atom-congruence:
  assumes
    term.is-ground (term₁ ·t γ)
    term.is-ground (term₂ ·t γ)
    (term.to-ground (γ var), term.to-ground update) ∈ I
     (term.to-ground (term₁ ·t γ(var := update)), term.to-ground (term₂ ·t γ(var
:= update))) ∈ I
  shows
    (term.to-ground (term₁ ·t γ), term.to-ground (term₂ ·t γ)) ∈ I
  using assms
  by (metis interpretation-term-congruence sym symE)

lemma interpretation-atom-congruence':
  assumes
    term.is-ground (term₁ ·t γ)
    term.is-ground (term₂ ·t γ)
    (term.to-ground (γ var), term.to-ground update) ∈ I
     (term.to-ground (term₁ ·t γ(var := update)), term.to-ground (term₂ ·t γ(var
:= update))) ∉ I
  shows
    (term.to-ground (term₁ ·t γ), term.to-ground (term₂ ·t γ)) ∉ I
  using assms
  by (metis interpretation-term-congruence' sym symE)

lemma interpretation-literal-congruence:
  assumes
    literal.is-ground (literal ·l γ)
    upair ` I ⊫l term.to-ground (Var var ·t γ) ≈ term.to-ground update
    upair ` I ⊫l literal.to-ground (literal ·l γ(var := update))
  shows
    upair ` I ⊫l literal.to-ground (literal ·l γ)
proof(cases literal)
  case (Pos atom)

  have atom.to-ground (atom ·a γ) ∈ upair ` I
```

```
  proof(cases atom)
    case (Upair term₁ term₂)
   then have term-groundings: term.is-ground (term₁ ·t γ) term.is-ground (term₂
·t γ)
      using Pos assms
      by clause-auto

    have (term.to-ground (γ var), term.to-ground update) ∈ I
      using sym assms by auto

    moreover have
      (term.to-ground (term₁ ·t γ(var := update)), term.to-ground (term₂ ·t γ(var
:= update))) ∈ I
      using assms Pos Upair
      unfolding literal.to-ground-def atom.to-ground-def
      by(auto simp: subst-atom sym subst-literal)

    ultimately show ?thesis
      using interpretation-atom-congruence[OF term-groundings]
      by (simp add: Upair sym subst-atom atom.to-ground-def)
  qed

  with Pos show ?thesis
    by (metis ground-atom-in-ground-literal(1) subst-literal(1) true-lit-simps(1))
next
  case (Neg atom)

  have atom.to-ground (atom ·a γ) ∉ upair ‘ I
  proof(cases atom)
    case (Upair term₁ term₂)
   then have term-groundings: term.is-ground (term₁ ·t γ) term.is-ground (term₂
·t γ)
      using Neg assms
      by clause-auto

    have (term.to-ground (γ var), term.to-ground update) ∈ I
      using sym assms by auto

    moreover have
      (term.to-ground (term₁ ·t γ(var := update)), term.to-ground (term₂ ·t γ(var
:= update))) ∉ I
      using assms Neg Upair
      unfolding literal.to-ground-def atom.to-ground-def
      by (simp add: sym subst-literal(2) subst-atom)

    ultimately show ?thesis
      using interpretation-atom-congruence'[OF term-groundings]
      by (simp add: Upair sym subst-atom atom.to-ground-def)
  qed
```

```
  then show ?thesis
   by (metis Neg ground-atom-in-ground-literal(2) subst-literal(2) true-lit-simps(2))
qed

lemma interpretation-clause-congruence:
  assumes
    clause.is-ground (clause · γ)
    upair ‘ I ⊫l term.to-ground (Var var ·t γ) ≈ term.to-ground update
    upair ‘ I ⊫ clause.to-ground (clause · γ(var := update))
  shows
    upair ‘ I ⊫ clause.to-ground (clause · γ)
  using assms
proof(induction clause)
  case empty
  then show ?case
    by clause-simp
next
  case (add literal clause′)

  have clause′-grounding: clause.is-ground (clause′ · γ)
    by (metis add.prems(1) clause-is-ground-add-mset subst-clause-add-mset)

  show ?case
  proof(cases upair ‘ I ⊫ clause.to-ground (clause′ · γ(var := update)))
    case True
    show ?thesis
      using add(1)[OF clause′-grounding assms(2) True]
      unfolding subst-clause-add-mset clause.to-ground-def
      by simp
  next
    case False
    then have upair ‘ I ⊫l literal.to-ground (literal ·l γ(var := update))
      using add.prems
     by (metis (no-types, lifting) image-mset-add-mset subst-clause-add-mset clause.to-ground-def
true-cls-add-mset)

    then have upair ‘ I ⊫l literal.to-ground (literal ·l γ)
      using interpretation-literal-congruence add.prems
      by (metis clause-is-ground-add-mset subst-clause-add-mset)

    then show ?thesis
      by (simp add: subst-clause-add-mset clause.to-ground-def)
  qed
qed

end
end
```

5.2 Renaming

context
  fixes ϱ :: ($'f$, $'v$) *subst*
  assumes *renaming*: *term-subst.is-renaming* ϱ
begin

lemma *renaming-vars-term*: *Var* ‘ *term.vars* (*term* ·*t* ϱ) = ϱ ‘ (*term.vars term*)
proof(*induction term*)
  case *Var*
  with *renaming* **show** *?case*
    unfolding *term-subst-is-renaming-iff*
    by (*metis Term.term.simps*(*17*) *eval-term.simps*(*1*) *image-empty image-insert is-VarE*)
next
  case (*Fun f terms*)

  have
    $\bigwedge$*term x*. ⟦*term* ∈ *set terms*; *x* ∈ *term.vars* (*term* ·*t* ϱ)⟧
      ⟹ *Var x* ∈ ϱ ‘ $\bigcup$ (*term.vars* ‘ *set terms*)
    using *Fun*
    by (*smt* (*verit*, *del-insts*) *UN-iff image-UN image-eqI*)

  moreover have
    $\bigwedge$*term x*. ⟦*term* ∈ *set terms*; *x* ∈ *term.vars term*⟧
      ⟹ ϱ *x* ∈ *Var* ‘ ($\bigcup$*x'* ∈ *set terms*. *term.vars* (*x'* ·*t* ϱ))
    using *Fun*
    by (*smt* (*verit*, *del-insts*) *UN-iff image-UN image-eqI*)

  ultimately show *?case*
    by *auto*
qed

lemma *renaming-vars-atom*: *Var* ‘ *atom.vars* (*atom* ·*a* ϱ) = ϱ ‘ *atom.vars atom*
  unfolding *atom.vars-def atom.subst-def*
  by(*cases atom*)(*auto simp*: *image-Un renaming-vars-term*)

lemma *renaming-vars-literal*: *Var* ‘ *literal.vars* (*literal* ·*l* ϱ) = ϱ ‘ *literal.vars literal*
  unfolding *literal.vars-def literal.subst-def*
  by(*cases literal*)(*auto simp*: *renaming-vars-atom*)

lemma *renaming-vars-clause*: *Var* ‘ *clause.vars* (*clause* · ϱ) = ϱ ‘ *clause.vars clause*
  using *renaming-vars-literal*
  by(*induction clause*)(*clause-auto simp*: *image-Un empty-clause-is-ground*)

lemma *surj-the-inv*: *surj* (λ*x*. *the-inv* ϱ (*Var x*))
  by (*metis is-Var-def renaming surj-def term-subst-is-renaming-iff the-inv-f-f*)

end

lemma *needed*: *surj g* ⟹ *infinite* {*x*. *f x* = *ty*} ⟹ *infinite* {*x*. *f* (*g x*) = *ty*}
by (*smt* (*verit*) *UNIV-I finite-imageI image-iff mem-Collect-eq rev-finite-subset subset-eq*)

lemma *obtain-ground-fun*:
assumes *term.is-ground t*
obtains *f ts* **where** *t* = *Fun f ts*
using *assms*
by(*cases t*) *auto*

lemma *vars-term-subst*: *term.vars* (*t* ·*t* σ) ⊆ *term.vars t* ∪ *range-vars* σ
by (*meson Diff-subset order-refl subset-trans sup.mono vars-term-subst-apply-term-subset*)

lemma *vars-term-imgu* [*clause-intro*]:
assumes *term-subst.is-imgu* μ {{*s*, *s*′}}
shows *term.vars* (*t* ·*t* μ) ⊆ *term.vars t* ∪ *term.vars s* ∪ *term.vars s*′
using *range-vars-subset-if-is-imgu*[*OF assms*] *vars-term-subst*
by *fastforce*

lemma *vars-context-imgu* [*clause-intro*]:
assumes *term-subst.is-imgu* μ {{*s*, *s*′}}
shows *context.vars* (*c* $\cdot t_c$ μ) ⊆ *context.vars c* ∪ *term.vars s* ∪ *term.vars s*′
using *vars-term-imgu*[*OF assms*, *of c*⟨*s*⟩]
by *simp*

lemma *vars-atom-imgu* [*clause-intro*]:
assumes *term-subst.is-imgu* μ {{*s*, *s*′}}
shows *atom.vars* (*a* ·*a* μ) ⊆ *atom.vars a* ∪ *term.vars s* ∪ *term.vars s*′
using *vars-term-imgu*[*OF assms*]
unfolding *atom.vars-def atom.subst-def*
by(*cases a*) *auto*

lemma *vars-literal-imgu* [*clause-intro*]:
assumes *term-subst.is-imgu* μ {{*s*, *s*′}}
shows *literal.vars* (*l* ·*l* μ) ⊆ *literal.vars l* ∪ *term.vars s* ∪ *term.vars s*′
using *vars-atom-imgu*[*OF assms*]
unfolding *literal.vars-def literal.subst-def set-literal-atm-of*
by (*metis* (*no-types*, *lifting*) *UN-insert ccSUP-empty literal.map-sel sup-bot.right-neutral*)

lemma *vars-clause-imgu* [*clause-intro*]:
assumes *term-subst.is-imgu* μ {{*s*, *s*′}}
shows *clause.vars* (*c* · μ) ⊆ *clause.vars c* ∪ *term.vars s* ∪ *term.vars s*′
using *vars-literal-imgu*[*OF assms*]
unfolding *clause.vars-def clause.subst-def*
by *blast*

end
theory *Fun-Extra*
imports *Main HOL−Library.Countable-Set HOL−Cardinals.Cardinals*

```
begin

lemma obtain-bij-betw-endo:
  assumes finite domain finite img card img = card domain
  obtains f
  where bij-betw f domain img ⋀x. x ∉ domain ⟹ f x = x
proof−
  obtain f′ where bij-f′: bij-betw f′ domain img
    using assms(3) bij-betw-iff-card[OF assms(1, 2)]
    by presburger

  let ?f = λx. if x ∈ domain then f′ x else x

  have bij-betw ?f domain img
    using bij-f′
    unfolding bij-betw-def inj-on-def
    by simp

  moreover have ⋀x. x ∉ domain ⟹ ?f x = x
    by simp

  ultimately show ?thesis
    using that
    unfolding inj-def
    by blast
qed

lemma obtain-bij-betw-inj-endo:
  assumes finite domain finite img card img = card domain domain ∩ img = {}
  obtains f
  where
    bij-betw f domain img
    bij-betw f img domain
    ⋀x. x ∉ domain ⟹ x ∉ img ⟹ f x = x
    inj f
proof−
  obtain f′ where bij-f′: bij-betw f′ domain img
    using assms(3) bij-betw-iff-card[OF assms(1, 2)]
    by auto

  obtain f′′ where bij-f′′: bij-betw f′′ img domain
    using assms(3) bij-betw-iff-card[OF assms(2, 1)]
    by blast

  let ?f = λx. if x ∈ domain then f′ x else if x ∈ img then f′′ x else x

  have bij-betw ?f domain img
    using bij-f′ bij-f′′
    unfolding bij-betw-def inj-on-def
```

by *auto*

moreover have *bij-betw ?f img domain*
using *bij-f′ bij-f″*
unfolding *bij-betw-def inj-on-def*
by (*smt* (*verit*) *assms*(*4*) *disjoint-iff image-cong*)

moreover have $\bigwedge x.\ x \notin domain \Longrightarrow x \notin img \ \Longrightarrow ?f\ x = x$
by *simp*

ultimately show *?thesis*
using *that*
unfolding *inj-def*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*4*) *bij-betw-iff-bijections disjoint-iff*)
qed

lemma *obtain-inj-on*:
assumes *finite domain infinite image-subset*
obtains *f*
where
inj-on (*f* :: *′a* ⇒ *′b*) *domain*
f ‘ *domain* ⊆ *image-subset*
proof−
let *?image* = *UNIV* :: *′b set*
let *?domain-size* = *card domain*

have *image-subset* ⊆ *?image*
by *simp*

obtain *image-subset′* **where**
image-subset′ ⊆ *image-subset* **and**
card image-subset′ = *?domain-size* **and**
finite image-subset′
by (*meson assms*(*2*) *infinite-arbitrarily-large*)

then obtain *f* **where** *bij*: *bij-betw f domain image-subset′*
by (*metis assms*(*1*) *bij-betw-iff-card*)

then have *inj*: *inj-on f domain*
using *bij-betw-def* **by** *auto*

with *bij* **have** *f* ‘ *domain* ⊆ *image-subset*
by (*simp add*: ‹*image-subset′* ⊆ *image-subset*› *bij-betw-def*)

with *inj* **show** *?thesis*
using *that*
by *blast*
qed

corollary *obtain-inj-on′*:
  assumes *finite domain infinite* (*UNIV* :: *′b set*)
  obtains *f*
  where *inj-on* (*f* :: *′a* ⇒ *′b*) *domain*
  using *obtain-inj-on*[*OF assms*]
  by *auto*

corollary *obtain-inj*:
  assumes *finite* (*UNIV* :: *′a set*) *infinite* (*UNIV* :: *′b set*)
  obtains *f*
  where *inj* (*f* :: *′a* ⇒ *′b*)
  using *obtain-inj-on*[*OF assms*]
  by *auto*

corollary *obtain-inj′*:
  assumes *finite* (*UNIV* :: *′a set*) *infinite image-subset*
  obtains *f*
  where *inj* (*f* :: *′a* ⇒ *′b*) *f* ‘ *domain* ⊆ *image-subset*
  using *obtain-inj-on*[*OF assms*]
  by (*metis image-subset-iff range-subsetD*)

lemma *obtain-inj-endo*:
  assumes *finite domain infinite image-subset*
  obtains *f* :: *′a* ⇒ *′a*
  where *inj f f* ‘ *domain* ⊆ *image-subset*
proof−
  let *?image* = *UNIV* :: *′b set*
  let *?domain-size* = *card domain*

  have *image-subset* ⊆ *?image*
    by *simp*

  obtain *image-subset′* **where** *image-subset′*:
    image-subset′ ⊆ *image-subset* − *domain*
    finite image-subset′
    card image-subset′ = *?domain-size*
    using *finite-Diff2*[*OF assms*(*1*)] *infinite-arbitrarily-large assms*(*2*)
    by *metis*

  then have *domain-image-subset′-distinct*: *domain* ∩ *image-subset′* = {}
    by *blast*

  obtain *image-subset′-inv domain-inv* **where** *xy*:
    image-subset′-inv = *UNIV* − *image-subset′*
    domain-inv = *UNIV* − *domain*
    by *blast*

  obtain *f* **where**
      bij-betw f domain image-subset′

bij-betw f image-subset′ domain
inj f
using *obtain-bij-betw-inj-endo*[*OF*
assms(*1*) *image-subset′*(*2*) *image-subset′*(*3*) *domain-image-subset′-distinct*
]
by *metis*

moreover then have *f ‘ domain* ⊆ *image-subset*
by (*metis Diff-subset bij-betw-def image-subset′*(*1*) *order-trans*)

ultimately show *?thesis*
using *that*
by *blast*
qed

abbreviation *surj-on* **where**
surj-on domain f ≡ (∀ *y*. ∃ *x* ∈ *domain*. *y* = *f x*)

lemma *surj-on-alternative*: *surj-on domain f* ⟷ *f ‘ domain* = *UNIV*
by *auto*

lemma *obtain-surj-on-nat*:
assumes *infinite domain*
obtains *f* :: ′*a* ⇒ *nat* **where** *surj-on domain f*
proof−
obtain *subdomain* **where**
subdomain: *infinite subdomain countable subdomain subdomain* ⊆ *domain*
using *infinite-countable-subset′*[*OF assms*]
by *blast*

then obtain *f* :: ′*a* ⇒ *nat* **where** *surj-on subdomain f*
by (*metis to-nat-on-surj*)

then have *surj-on domain f*
using *subdomain*(*3*)
by (*meson subset-iff*)

then show *?thesis*
using *that*
by *blast*
qed

lemma *obtain-surj-on*:
assumes *infinite domain*
obtains *f* :: ′*a* ⇒ ′*b* :: *countable* **where** *surj-on domain f*
proof−
obtain *f′* :: ′*a* ⇒ *nat*
where *f′*: *surj-on domain f′*
using *obtain-surj-on-nat*[*OF assms*]

by *blast*

let *?f* = (*from-nat* :: *nat* ⇒ *'b*) ∘ *f'*

have *f*: ∀ *y*. ∃ *x*∈*domain*. *y* = *?f x*
using *f'*
unfolding *comp-def*
by (*metis from-nat-to-nat*)

show *?thesis*
using *that*[*OF f*].
qed

lemma *partitions*:
assumes *infinite* (*UNIV* :: *'x set*)
obtains *A B* **where**
|*A*| =*o* |*B*|
|*A*| =*o* |*UNIV* :: *'x set*|
A ∩ *B* = {}
A ∪ *B* = (*UNIV* :: *'x set*)
proof−
obtain *f* :: *'x* + *'x* ⇒ *'x* **where** *f*: *bij f*
by (*meson Plus-infinite-bij-betw-types assms bij-betw-inv one-type-greater*)

define *A* :: *'x set* **where** *A* ≡ *f* ‘ *range Inl*
define *B* :: *'x set* **where** *B* ≡ *f* ‘ *range Inr*

have *A* ∩ *B* = {}
unfolding *A-def B-def*
by (*smt* (*verit*, *best*) *Inl-Inr-False UNIV-I bij-betw-iff-bijections disjoint-iff f imageE*)

moreover have *A* ∪ *B* = *UNIV*
unfolding *A-def B-def*
by (*metis UNIV-sum bij-is-surj f image-Un*)

moreover have *Inl*: |*Inl* ‘ (*UNIV* :: *'x set*)| =*o* |*UNIV* :: *'x set*|
by (*meson bij-betw-imageI card-of-ordIsoI inj-Inl ordIso-symmetric*)

have *Inr*: |*Inr* ‘ (*UNIV* :: *'x set*)| =*o* |*UNIV* :: *'x set*|
by (*meson bij-betw-imageI card-of-ordIsoI inj-Inr ordIso-symmetric*)

have |*A*| =*o* |*UNIV* :: *'x set*|
unfolding *A-def*
using *f*
unfolding *bij-betw-def*
by (*metis Inl Int-UNIV-left bij-betw-imageI bij-betw-inv card-of-ordIsoI inj-on-Int ordIso-transitive*)

moreover have $|B| =o$ $|UNIV :: 'x\ set|$
using *f*
unfolding *B-def bij-betw-def*
by (*meson UNIV-I bij-betw-imageI card-of-ordIsoI inj-Inr inj-on-def ordIso-symmetric ordIso-transitive*)

ultimately show *?thesis*
using *that*
by (*meson ordIso-symmetric ordIso-transitive*)
qed

end
theory *First-Order-Type-System*
imports *First-Order-Clause Fun-Extra*
begin

type-synonym $('f,\ 'ty)$ *fun-types* = $'f \Rightarrow 'ty\ list \times 'ty$
type-synonym $('v,\ 'ty)$ *var-types* = $'v \Rightarrow 'ty$

inductive *has-type* :: $('f,\ 'ty)$ *fun-types* $\Rightarrow$ $('v,\ 'ty)$ *var-types* $\Rightarrow$ $('f,'v)$ *term* $\Rightarrow$ $'ty \Rightarrow bool$
for $\mathcal{F}$ $\mathcal{V}$ **where**
Var: $\mathcal{V}\ x = \tau \Longrightarrow$ *has-type* $\mathcal{F}$ $\mathcal{V}$ (*Var x*) τ
| *Fun*: $\mathcal{F}\ f = (\tau s,\ \tau) \Longrightarrow$ *has-type* $\mathcal{F}$ $\mathcal{V}$ (*Fun f ts*) τ

inductive *welltyped* :: $('f,\ 'ty)$ *fun-types* $\Rightarrow$ $('v,\ 'ty)$ *var-types* $\Rightarrow$ $('f,'v)$ *term* $\Rightarrow$ $'ty \Rightarrow bool$
for $\mathcal{F}$ $\mathcal{V}$ **where**
Var: $\mathcal{V}\ x = \tau \Longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var x*) τ
| *Fun*: $\mathcal{F}\ f = (\tau s,\ \tau) \Longrightarrow$ *list-all2* (*welltyped* $\mathcal{F}$ $\mathcal{V}$) *ts* $\tau s \Longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Fun f ts*) τ

lemma *has-type-right-unique*: *right-unique* (*has-type* $\mathcal{F}$ $\mathcal{V}$)
proof (*rule right-uniqueI*)
fix $t\ \tau_1\ \tau_2$
assume *has-type* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau_1$ **and** *has-type* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau_2$
thus $\tau_1 = \tau_2$
by (*auto elim*!: *has-type.cases*)
qed

lemma *welltyped-right-unique*: *right-unique* (*welltyped* $\mathcal{F}$ $\mathcal{V}$)
proof (*rule right-uniqueI*)
fix $t\ \tau_1\ \tau_2$
assume *welltyped* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau_1$ **and** *welltyped* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau_2$
thus $\tau_1 = \tau_2$
by (*auto elim*!: *welltyped.cases*)

qed

definition *has-type*$_a$ **where**
has-type$_a$ $\mathcal{F}$ $\mathcal{V}$ $A \longleftrightarrow (\exists\,\tau.\ \forall\, t \in$ *set-uprod* $A.$ *has-type* $\mathcal{F}$ $\mathcal{V}$ t $\tau)$

definition *welltyped*$_a$ **where**
[*clause-simp*]: *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ $A \longleftrightarrow (\exists\,\tau.\ \forall\, t \in$ *set-uprod* $A.$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t $\tau)$

definition *has-type*$_l$ **where**
has-type$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \longleftrightarrow$ *has-type*$_a$ $\mathcal{F}$ $\mathcal{V}$ (*atm-of* L)

definition *welltyped*$_l$ **where**
[*clause-simp*]: *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ (*atm-of* L)

definition *has-type*$_c$ **where**
has-type$_c$ $\mathcal{F}$ $\mathcal{V}$ $C \longleftrightarrow (\forall\, L \in\#\ C.$ *has-type*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L)$

definition *welltyped*$_c$ **where**
welltyped$_c$ $\mathcal{F}$ $\mathcal{V}$ $C \longleftrightarrow (\forall\, L \in\#\ C.$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L)$

definition *has-type*$_{cs}$ **where**
has-type$_{cs}$ $\mathcal{F}$ $\mathcal{V}$ $N \longleftrightarrow (\forall\, C \in N.$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ $C)$

definition *welltyped*$_{cs}$ **where**
welltyped$_{cs}$ $\mathcal{F}$ $\mathcal{V}$ $N \longleftrightarrow (\forall\, C \in N.$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ $C)$

definition *has-type*$_\sigma$ **where**
has-type$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ $\sigma \longleftrightarrow (\forall\, t\ \tau.$ *has-type* $\mathcal{F}$ $\mathcal{V}$ t $\tau \longrightarrow$ *has-type* $\mathcal{F}$ $\mathcal{V}$ $(t \cdot t\ \sigma)$ $\tau)$

definition *has-type*$_\sigma{}'$ **where**
has-type$_\sigma{}'$ $\mathcal{F}$ $\mathcal{V}$ $\sigma \longleftrightarrow (\forall\, x.$ *has-type* $\mathcal{F}$ $\mathcal{V}$ $(\sigma\ x)$ $(\mathcal{V}\ x))$

definition *welltyped*$_\sigma$ **where**
welltyped$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ $\sigma \longleftrightarrow (\forall\, x.$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ $(\sigma\ x)$ $(\mathcal{V}\ x))$

lemma *welltyped*$_\sigma$*-Var*[*simp*]: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ *Var*
unfolding *welltyped*$_\sigma$*-def*
by (*simp add*: *welltyped.intros*)

definition *welltyped*$_\sigma$*-on* **where**
welltyped$_\sigma$*-on* X $\mathcal{F}$ $\mathcal{V}$ $\sigma \longleftrightarrow (\forall\, x \in X.$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ $(\sigma\ x)$ $(\mathcal{V}\ x))$

lemma *welltyped*$_\sigma$*-welltyped*$_\sigma$*-on*:
welltyped$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ $\sigma =$ *welltyped*$_\sigma$*-on* *UNIV* $\mathcal{F}$ $\mathcal{V}$ σ
unfolding *welltyped*$_\sigma$*-def* *welltyped*$_\sigma$*-on-def*
by *blast*

lemma *welltyped*$_\sigma$*-on-subset*:
assumes *welltyped*$_\sigma$*-on* Y $\mathcal{F}$ $\mathcal{V}$ σ $X \subseteq Y$

shows *welltyped*$_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}$ σ
using *assms*
unfolding *welltyped*$_\sigma$*-on-def*
by *blast*

definition *welltyped*$_\sigma$*'* **where**
welltyped$_\sigma$*'* $\mathcal{F}$ $\mathcal{V}$ $\sigma \longleftrightarrow$ $(\forall t\ \tau.$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau \longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ $(t \cdot t\ \sigma)\ \tau)$

lemma *has-type*$_c$*-add-mset* [*clause-simp*]:
has-type$_c$ $\mathcal{F}$ $\mathcal{V}$ (*add-mset L C*) $\longleftrightarrow$ *has-type*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \land$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ C
by (*simp add*: *has-type*$_c$*-def*)

lemma *welltyped*$_c$*-add-mset* [*clause-simp*]:
welltyped$_c$ $\mathcal{F}$ $\mathcal{V}$ (*add-mset L C*) $\longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \land$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ C
by (*simp add*: *welltyped*$_c$*-def*)

lemma *has-type*$_c$*-plus* [*clause-simp*]:
has-type$_c$ $\mathcal{F}$ $\mathcal{V}$ $(C + D) \longleftrightarrow$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ $C \land$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ D
by (*auto simp*: *has-type*$_c$*-def*)

lemma *welltyped*$_c$*-plus* [*clause-simp*]:
welltyped$_c$ $\mathcal{F}$ $\mathcal{V}$ $(C + D) \longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ $C \land$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ D
by (*auto simp*: *welltyped*$_c$*-def*)

lemma *has-type*$_\sigma$*-has-type*:
assumes *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ *has-type* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau$
shows *has-type* $\mathcal{F}$ $\mathcal{V}$ $(t \cdot t\ \sigma)\ \tau$
using *assms*
unfolding *has-type*$_\sigma$*-def*
by *blast*

lemma *welltyped*$_\sigma$*-welltyped*:
assumes *welltyped*$_\sigma$: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ $(t \cdot t\ \sigma)\ \tau \longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau$
proof(*rule iffI*)
assume *welltyped* $\mathcal{F}$ $\mathcal{V}$ $(t \cdot t\ \sigma)\ \tau$
thus *welltyped* $\mathcal{F}$ $\mathcal{V}$ $t\ \tau$
proof(*induction* $t \cdot t\ \sigma\ \tau$ *arbitrary*: t *rule*: *welltyped.induct*)
case (*Var* $x\ \tau$)
then obtain x' **where** t: $t =$ *Var* x'
by (*metis subst-apply-eq-Var*)

have *welltyped* $\mathcal{F}$ $\mathcal{V}$ t $(\mathcal{V}\ x')$
unfolding t
by (*simp add*: *welltyped.Var*)

have *welltyped* $\mathcal{F}$ $\mathcal{V}$ t $(\mathcal{V}\ x)$
using *Var welltyped*$_\sigma$
unfolding t *welltyped*$_\sigma$*-def*

```
    by (metis eval-term.simps(1) welltyped.Var right-uniqueD welltyped-right-unique)

    then have 𝒱-x': τ = 𝒱 x'
      using Var welltyped_σ
      unfolding welltyped_σ-def t
      by (metis welltyped.Var right-uniqueD welltyped-right-unique t)

    show ?case
      unfolding t 𝒱-x'
      by (simp add: welltyped.Var)
  next
    case (Fun f τs τ ts)
    show ?case
    proof(cases t)
      case (Var x)
      from Fun show ?thesis
        using welltyped_σ
        unfolding welltyped_σ-def Var
        by (metis (no-types, opaque-lifting) eval-term.simps(1) prod.sel(2)
            term.distinct(1) term.inject(2) welltyped.simps)
    next
      case Fun_t: Fun
      with Fun show ?thesis
        by (simp add: welltyped.simps list.rel-map(1) list-all2-mono)
    qed
  qed
next
  assume welltyped ℱ 𝒱 t τ
  thus welltyped ℱ 𝒱 (t ·t σ) τ
  proof(induction t τ rule: welltyped.induct)
    case Var_t: (Var x τ)
    then show ?case
    proof(cases Var x ·t σ)
      case Var
      then show ?thesis
        using welltyped_σ
        unfolding welltyped_σ-def
        by (metis Var_t.hyps eval-term.simps(1))
    next
      case Fun
      then show ?thesis
        using welltyped_σ
        unfolding welltyped_σ-def
        by (metis Var_t.hyps eval-term.simps(1))
    qed
  next
    case (Fun f τs τ ts)
    then show ?case
      using assms list-all2-mono
```

```
    unfolding welltyped_σ-def
    by (smt (verit, ccfv-SIG) eval-term.simps(2) welltyped.simps list.rel-map(1))
  qed
qed

lemma has-type_σ-has-type_a:
  assumes has-type_σ 𝓕 𝓥 σ has-type_a 𝓕 𝓥 a
  shows has-type_a 𝓕 𝓥 (a ·a σ)
  using assms has-type_σ-has-type
  unfolding has-type_a-def atom.subst-def
  by(cases a) fastforce

lemma welltyped_σ-welltyped_a:
  assumes welltyped_σ: welltyped_σ 𝓕 𝓥 σ
  shows welltyped_a 𝓕 𝓥 (a ·a σ) ⟷ welltyped_a 𝓕 𝓥 a
  using welltyped_σ-welltyped[OF welltyped_σ]
  unfolding welltyped_a-def atom.subst-def
  by(cases a) simp

lemma has-type_σ-has-type_l:
  assumes has-type_σ 𝓕 𝓥 σ has-type_l 𝓕 𝓥 l
  shows has-type_l 𝓕 𝓥 (l ·l σ)
  using assms has-type_σ-has-type_a
  unfolding has-type_l-def literal.subst-def
  by(cases l) auto

lemma welltyped_σ-welltyped_l:
  assumes welltyped_σ: welltyped_σ 𝓕 𝓥 σ
  shows welltyped_l 𝓕 𝓥 (l ·l σ) ⟷ welltyped_l 𝓕 𝓥 l
  using welltyped_σ-welltyped_a[OF welltyped_σ]
  unfolding welltyped_l-def literal.subst-def
  by(cases l) auto

lemma has-type_σ-has-type_c:
  assumes has-type_σ 𝓕 𝓥 σ has-type_c 𝓕 𝓥 c
  shows has-type_c 𝓕 𝓥 (c · σ)
  using assms has-type_σ-has-type_l
  unfolding has-type_c-def clause.subst-def
  by blast

lemma welltyped_σ-on-welltyped:
  assumes wt: welltyped_σ-on (term.vars t) 𝓕 𝓥 σ
  shows welltyped 𝓕 𝓥 (t ·t σ) τ ⟷ welltyped 𝓕 𝓥 t τ
proof(rule iffI)
  assume welltyped 𝓕 𝓥 (t ·t σ) τ
  thus welltyped 𝓕 𝓥 t τ
    using wt
  proof(induction t ·t σ τ  arbitrary: t rule: welltyped.induct)
    case (Var x τ)
```

```
    then obtain x' where t: t = Var x'
      by (metis subst-apply-eq-Var)

    have welltyped 𝓕 𝓥 t (𝓥 x')
      unfolding t
      by (simp add: welltyped.Var)

    have welltyped 𝓕 𝓥 t (𝓥 x)
      using Var
      unfolding t welltyped_σ-on-def
      by (auto intro: welltyped.Var elim: welltyped.cases)

    then have 𝓥-x': τ = 𝓥 x'
      using Var
      unfolding welltyped_σ-def  t
      by (metis welltyped.Var right-uniqueD welltyped-right-unique t)

    show ?case
      unfolding t 𝓥-x'
      by (simp add: welltyped.Var)
  next
    case (Fun f τs τ ts)
    show ?case
    proof(cases t)
      case (Var x)
      from Fun show ?thesis
        using Fun
        unfolding welltyped_σ-def Var
        by (simp add: welltyped.simps welltyped_σ-on-def)
    next
      case Fun_t: (Fun f' ts')
      hence f = f' and ts = map (λt. t ·t σ) ts'
        using ‹Fun f ts = t ·t σ› by simp-all

      show ?thesis
        unfolding Fun_t
      proof (rule welltyped.Fun)
        show 𝓕 f' = (τs, τ)
          using Fun.hyps ‹f = f'› by argo
      next
        show list-all2 (welltyped 𝓕 𝓥) ts' τs
        proof (rule list.rel-mono-strong)
          show list-all2 (λx x2. welltyped 𝓕 𝓥 (x ·t σ) x2 ∧
              (∀xa. x ·t σ = xa ·t σ ⟶ welltyped_σ-on (term.vars xa) 𝓕 𝓥 σ ⟶
welltyped 𝓕 𝓥 xa x2))
            ts' τs
            using Fun.hyps
            unfolding ‹ts = map (λt. t ·t σ) ts'› list.rel-map
            by argo
```

```
        next
          fix t′ τ′
          assume
            t′ ∈ set ts′ and
            τ′ ∈ set τs and
            welltyped ℱ 𝒱 (t′ ·t σ) τ′ ∧
              (∀ xa. t′ ·t σ = xa ·t σ ⟶ welltyped_σ-on (term.vars xa) ℱ 𝒱 σ ⟶
                welltyped ℱ 𝒱 xa τ′)
          thus welltyped ℱ 𝒱 t′ τ′
            using Fun.prems Fun.hyps
            by (simp add: Fun_t welltyped_σ-on-def)
        qed
      qed
    qed
  qed
next
  assume welltyped ℱ 𝒱 t τ
  thus welltyped ℱ 𝒱 (t ·t σ) τ
    using wt
  proof(induction t τ  rule: welltyped.induct)
    case Var_t: (Var x τ)
    thus ?case
      by (cases Var x ·t σ) (simp-all add: welltyped_σ-on-def)
  next
    case (Fun f τs τ ts)

    show ?case
      unfolding eval-term.simps
    proof (rule welltyped.Fun)
      show ℱ f = (τs, τ)
        using Fun by argo
    next
      show list-all2 (welltyped ℱ 𝒱) (map (λs. s ·t σ) ts) τs
        unfolding list.rel-map
        using Fun.IH
      proof (rule list.rel-mono-strong)
        fix t and τ′
        assume
          t ∈ set ts and
          τ′ ∈ set τs and
          welltyped ℱ 𝒱 t τ′ ∧ (welltyped_σ-on (term.vars t) ℱ 𝒱 σ ⟶ welltyped ℱ
𝒱 (t ·t σ) τ′)
        thus welltyped ℱ 𝒱 (t ·t σ) τ′
          using Fun.prems
          by (simp add: welltyped_σ-on-def)
      qed
    qed
  qed
qed
```

```
lemma welltypedσ-on-welltypeda:
  assumes wt: welltypedσ-on (atom.vars A) 𝓕 𝓥 σ
  shows welltypeda 𝓕 𝓥 (A ·a σ) ⟷ welltypeda 𝓕 𝓥 A
proof (cases A)
  case (Upair t t′)

  have welltypedσ-on (term.vars t) 𝓕 𝓥 σ welltypedσ-on (term.vars t′) 𝓕 𝓥 σ
    using wt unfolding Upair by (simp-all add: welltypedσ-on-def atom.vars-def)

  hence (∃τ. welltyped 𝓕 𝓥 (t ·t σ) τ ∧ welltyped 𝓕 𝓥 (t′ ·t σ) τ) =
    (∃τ. welltyped 𝓕 𝓥 t τ ∧ welltyped 𝓕 𝓥 t′ τ)
    using welltypedσ-on-welltyped by metis

  thus ?thesis
    using Upair
    by (simp add: atom.subst-def welltypeda-def)
qed

lemma welltypedl-iff-welltypeda: welltypedl 𝓕 𝓥 L ⟷ welltypeda 𝓕 𝓥 (atm-of L)
  by (cases L) (simp-all add: welltypedl-def)

lemma welltypedσ-on-welltypedl:
  assumes wt: welltypedσ-on (literal.vars L) 𝓕 𝓥 σ
  shows welltypedl 𝓕 𝓥 (L ·l σ) ⟷ welltypedl 𝓕 𝓥 L
  unfolding welltypedl-iff-welltypeda subst-literal
proof (rule welltypedσ-on-welltypeda)
  have atom.vars (atm-of L) = literal.vars L
    by (cases L) clause-auto
  thus welltypedσ-on (atom.vars (atm-of L)) 𝓕 𝓥 σ
    using wt
    by simp
qed

lemma welltypedσ-on-welltypedc:
  assumes wt: welltypedσ-on (clause.vars C) 𝓕 𝓥 σ
  shows welltypedc 𝓕 𝓥 (C · σ) ⟷ welltypedc 𝓕 𝓥 C
proof −
  have welltypedl 𝓕 𝓥 (L ·l σ) ⟷ welltypedl 𝓕 𝓥 L if L ∈# C for L
  proof (rule welltypedσ-on-welltypedl)
    have literal.vars L ⊆ clause.vars C
      using ‹L ∈# C›
      by (simp add: UN-upper clause.vars-def)
    thus welltypedσ-on (literal.vars L) 𝓕 𝓥 σ
      using wt welltypedσ-on-subset by metis
  qed

  thus ?thesis
    unfolding welltypedc-def clause.subst-def
```

by *simp*
qed

lemma $welltyped_\sigma$-$welltyped_c$:
assumes $welltyped_\sigma$: $welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ \sigma$
shows $welltyped_c\ \mathcal{F}\ \mathcal{V}\ (c \cdot \sigma) \longleftrightarrow welltyped_c\ \mathcal{F}\ \mathcal{V}\ c$
using $welltyped_\sigma$-$welltyped_l$[*OF* $welltyped_\sigma$]
unfolding $welltyped_c$-*def clause.subst-def*
by *blast*

lemma *has-type*$_\kappa$:
assumes
κ-*type*: *has-type* $\mathcal{F}\ \mathcal{V}\ \kappa\langle t\rangle\ \tau_1$ **and**
t-type: *has-type* $\mathcal{F}\ \mathcal{V}\ t\ \tau_2$ **and**
t'-*type*: *has-type* $\mathcal{F}\ \mathcal{V}\ t'\ \tau_2$
shows
has-type $\mathcal{F}\ \mathcal{V}\ \kappa\langle t'\rangle\ \tau_1$
using κ-*type*
proof(*induction* κ *arbitrary*: τ_1)
case *Hole*
then show *?case*
using *has-type-right-unique right-uniqueD* t'-*type t-type* **by** *fastforce*
next
case *More*
then show *?case*
by (*simp add*: *has-type.simps*)
qed

lemma *welltyped-subterm*:
assumes *welltyped* $\mathcal{F}\ \mathcal{V}$ (*Fun f ts*) τ
shows $\forall\, t \in set\ ts.\ \exists \tau'.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau'$
using *assms*
proof(*induction ts*)
case *Nil*
then show *?case*
by *simp*
next
case (*Cons a ts*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *Term.term.simps*(*4*) *in-set-conv-nth list-all2-conv-all-nth*

term.sel(*4*) *welltyped.simps*)
qed

lemma $welltyped_\kappa'$:
assumes *welltyped* $\mathcal{F}\ \mathcal{V}\ \kappa\langle t\rangle\ \tau$
shows $\exists \tau'.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau'$
using *assms*
proof(*induction* κ *arbitrary*: τ)

```
  case Hole
  then show ?case
    by auto
next
  case (More x1 x2 κ x4)
  then show ?case
    by (metis ctxt-apply-term.simps(2) in-set-conv-decomp welltyped-subterm)

qed


lemma welltyped_κ [clause-intro]:
  assumes
    κ-type: welltyped ℱ 𝒱 κ⟨t⟩ τ₁ and
    t-type: welltyped ℱ 𝒱 t τ₂ and
    t'-type: welltyped ℱ 𝒱 t' τ₂
  shows
    welltyped ℱ 𝒱 κ⟨t'⟩ τ₁
  using κ-type
proof (induction κ arbitrary: τ₁)
  case Hole
  then show ?case
    using t-type t'-type welltyped-right-unique[of ℱ, THEN right-uniqueD]
    by auto
next
  case (More f ss1 κ ss2)
  have welltyped ℱ 𝒱 (Fun f (ss1 @ κ⟨t⟩ # ss2)) τ₁
    using More.prems by simp
  hence welltyped ℱ 𝒱 (Fun f (ss1 @ κ⟨t'⟩ # ss2)) τ₁
  proof (cases ℱ 𝒱 Fun f (ss1 @ κ⟨t⟩ # ss2) τ₁ rule: welltyped.cases)
    case (Fun τs)
    show ?thesis
    proof (rule welltyped.Fun)
      show ℱ f = (τs, τ₁)
        using ‹ℱ f = (τs, τ₁)› .
    next
      show list-all2 (welltyped ℱ 𝒱) (ss1 @ κ⟨t'⟩ # ss2) τs
        using ‹list-all2 (welltyped ℱ 𝒱) (ss1 @ κ⟨t⟩ # ss2) τs›
        using More.IH
        by (smt (verit, del-insts) list-all2-Cons1 list-all2-append1 list-all2-lengthD)
    qed
  qed
  thus ?case
    by simp
qed

lemma has-type_σ-Var: has-type_σ ℱ 𝒱 Var
  unfolding has-type_σ-def
  by simp
```

lemma *welltyped-add-literal*:
assumes $welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ P' *welltyped* $\mathcal{F}$ $\mathcal{V}$ s_1 τ *welltyped* $\mathcal{F}$ $\mathcal{V}$ s_2 τ
shows $welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ (*add-mset* (s_1 !≈ s_2) P')
using *assms*
unfolding $welltyped_c$*-add-mset* $welltyped_l$*-def* $welltyped_a$*-def*
by *auto*

lemma *welltyped-*$\mathcal{V}$:
assumes
$\forall x \in$*term.vars t*. $\mathcal{V}\ x = \mathcal{V}'\ x$
welltyped $\mathcal{F}$ $\mathcal{V}$ *t* τ
shows
welltyped $\mathcal{F}$ $\mathcal{V}'$ *t* τ
using *assms*(*2*, *1*)
by(*induction rule*: *welltyped.induct*)(*auto simp*: *welltyped.simps list.rel-mono-strong*)

lemma *welltyped-subst-*$\mathcal{V}$:
assumes
$\forall x \in X.\ \mathcal{V}\ x = \mathcal{V}'\ x$
$\forall x \in X.$ *term.is-ground* (γ *x*)
shows
$welltyped_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}$ γ $\longleftrightarrow$ $welltyped_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}'$ γ
unfolding $welltyped_\sigma$*-on-def*
using *welltyped-*$\mathcal{V}$ *assms*
by (*metis empty-iff*)

lemma $welltyped_a$-$\mathcal{V}$:
assumes
$\forall x \in$*atom.vars a*. $\mathcal{V}\ x = \mathcal{V}'\ x$
$welltyped_a$ $\mathcal{F}$ $\mathcal{V}$ *a*
shows
$welltyped_a$ $\mathcal{F}$ $\mathcal{V}'$ *a*
using *assms*
unfolding $welltyped_a$*-def atom.vars-def*
by (*metis* (*full-types*) *UN-I welltyped-*$\mathcal{V}$)

lemma $welltyped_l$-$\mathcal{V}$:
assumes
$\forall x \in$ *literal.vars l*. $\mathcal{V}\ x = \mathcal{V}'\ x$
$welltyped_l$ $\mathcal{F}$ $\mathcal{V}$ *l*
shows
$welltyped_l$ $\mathcal{F}$ $\mathcal{V}'$ *l*
using *assms* $welltyped_a$-$\mathcal{V}$
unfolding $welltyped_l$*-def literal.vars-def set-literal-atm-of*
by *fastforce*

lemma $welltyped_c$-$\mathcal{V}$:
assumes

```
    ∀x∈ clause.vars c. 𝒱 x = 𝒱' x
    welltyped_c ℱ 𝒱 c
  shows
    welltyped_c ℱ 𝒱' c
  using assms welltyped_l-𝒱
  unfolding welltyped_c-def clause.vars-def
  by fastforce

lemma welltyped-renaming':
  assumes
    term-subst.is-renaming ϱ
    welltyped_σ typeof-fun 𝒱 ϱ
    welltyped typeof-fun (λx. 𝒱 (the-inv Var (ϱ x))) t τ
  shows welltyped typeof-fun 𝒱 (t ·t ϱ) τ
  using assms(3)
proof(induction rule: welltyped.induct)
  case (Var x τ)
  then show ?case
    using assms(1, 2)
    unfolding welltyped_σ-def
    by (metis comp-apply eval-term.simps(1) inj-on-Var
        term-subst-is-renaming-iff-ex-inj-fun-on-vars the-inv-f-f welltyped.Var)
next
  case (Fun f τs τ ts)
  then show ?case
   by (smt (verit, ccfv-SIG) assms(2) list-all2-mono welltyped.Fun welltyped_σ-welltyped)
qed

lemma welltyped_a-renaming':
  assumes
    term-subst.is-renaming ϱ
    welltyped_σ typeof-fun 𝒱 ϱ
    welltyped_a typeof-fun (λx. 𝒱 (the-inv Var (ϱ x))) a
  shows welltyped_a typeof-fun 𝒱 (a ·a ϱ)
  using welltyped-renaming'[OF assms(1,2)] assms(3)
  unfolding welltyped_a-def
  by(cases a)(auto simp: subst-atom)

lemma welltyped_l-renaming':
  assumes
    term-subst.is-renaming ϱ
    welltyped_σ typeof-fun 𝒱 ϱ
    welltyped_l typeof-fun (λx. 𝒱 (the-inv Var (ϱ x))) l
  shows welltyped_l typeof-fun 𝒱 (l ·l ϱ)
  using welltyped_a-renaming'[OF assms(1,2)] assms(3)
  unfolding welltyped_l-def subst-literal(3)
  by presburger

lemma welltyped_c-renaming':
```

assumes
term-subst.is-renaming ϱ
welltyped$_\sigma$ *typeof-fun* $\mathcal{V}$ ϱ
welltyped$_c$ *typeof-fun* $(\lambda x.\ \mathcal{V}$ (*the-inv Var* $(\varrho\ x)))$ *c*
shows *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ $(c \cdot \varrho)$
using *welltyped*$_l$*-renaming*′[*OF assms*(*1*,*2*)] *assms*(*3*)
unfolding *welltyped*$_c$*-def*
by (*simp add*: *clause.subst-def*)

definition *range-vars*′ :: (′*f*, ′*v*) *subst* ⇒ ′*v set* **where**
range-vars′ $\sigma = \bigcup$(*term.vars* ‘ *range* σ)

lemma *vars-term-range-vars*′:
assumes $x \in$ *term.vars* $(t \cdot t\ \sigma)$
shows $x \in$ *range-vars*′ σ
using *assms*
unfolding *range-vars*′*-def*
by(*induction t*) *auto*

context
fixes $\varrho\ \mathcal{V}\ \mathcal{V}'$
assumes
renaming: *term-subst.is-renaming* ϱ **and**
range-vars: $\forall x \in$ *range-vars*′ ϱ. $\mathcal{V}$ (*the-inv* ϱ (*Var x*)) $= \mathcal{V}'\ x$
begin

lemma *welltyped-renaming*: *welltyped* $\mathcal{F}\ \mathcal{V}\ t\ \tau \longleftrightarrow$ *welltyped* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
proof(*intro iffI*)
assume *welltyped* $\mathcal{F}\ \mathcal{V}\ t\ \tau$
then show *welltyped* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
proof(*induction rule*: *welltyped.induct*)
case (*Var x* τ)

obtain *y* **where** *y*: *Var x* $\cdot t\ \varrho =$ *Var y*
using *renaming*
by (*metis eval-term.simps*(*1*) *term.collapse*(*1*) *term-subst-is-renaming-iff*)

then have $y \in$ *range-vars*′ ϱ
using *vars-term-range-vars*′
by (*metis term.set-intros*(*3*))

then have $\mathcal{V}$ (*the-inv* ϱ (*Var y*)) $= \mathcal{V}'\ y$
by (*simp add*: *range-vars*)

moreover have (*the-inv* ϱ (*Var y*)) $= x$
using *y renaming*
unfolding *term-subst-is-renaming-iff*
by (*metis eval-term.simps*(*1*) *the-inv-f-f*)

ultimately have $\mathcal{V}'\ y = \tau$
using *Var*
by *argo*

then show *?case*
unfolding *y*
by(*rule welltyped.Var*)
next
case (*Fun f τs τ ts*)
then show *?case*
by (*smt* (*verit*, *ccfv-SIG*) *eval-term.simps*(*2*) *length-map list-all2-conv-all-nth*

nth-map welltyped.simps)
qed
next
assume *welltyped* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
then show *welltyped* $\mathcal{F}\ \mathcal{V}\ t\ \tau$
proof(*induction t arbitrary*: τ)
case (*Var x*)
then obtain *y* **where** *y*: *Var x* $\cdot t\ \varrho =$ *Var y*
using *renaming*
by (*metis eval-term.simps*(*1*) *term.collapse*(*1*) *term-subst-is-renaming-iff*)

then have $y \in$ *range-vars'* ϱ
using *vars-term-range-vars'*
by (*metis term.set-intros*(*3*))

then have $\mathcal{V}$ (*the-inv* ϱ (*Var y*)) $= \mathcal{V}'\ y$
by (*simp add*: *range-vars*)

moreover have (*the-inv* ϱ (*Var y*)) $= x$
using *y renaming*
unfolding *term-subst-is-renaming-iff*
by (*metis eval-term.simps*(*1*) *the-inv-f-f*)

moreover have $\mathcal{V}'\ y = \tau$
using *Var*
unfolding *y*
by (*meson right-uniqueD welltyped.Var welltyped-right-unique*)

ultimately have $\mathcal{V}\ x = \tau$
by *blast*

then show *?case*
by(*rule welltyped.Var*)
next
case (*Fun f ts*)
then show *?case*
by (*smt* (*verit*, *ccfv-SIG*) *eval-term.simps*(*2*) *list.rel-map*(*1*) *list.rel-mono-strong*

term.distinct(*1*) *term.inject*(*2*) *welltyped.simps*)
qed
qed

lemma *has-type-renaming*: *has-type* $\mathcal{F}$ $\mathcal{V}$ *t* τ $\longleftrightarrow$ *has-type* $\mathcal{F}$ $\mathcal{V}'$ (*t* ·*t* ϱ) τ
using *renaming range-vars*
proof(*cases t*)
case (*Var x1*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *comp-apply eval-term.simps*(*1*) *has-type.simps range-vars renaming*
term.distinct(*1*) *term.set-intros*(*3*) *term-subst-is-renaming-iff*
term-subst-is-renaming-iff-ex-inj-fun-on-vars the-inv-f-f vars-term-range-vars')
next
case (*Fun x21 x22*)
then show *?thesis*
by (*simp add*: *has-type.simps*)
qed

lemma $welltyped_\sigma$*-renaming-ground-subst*:
assumes $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}'$ γ $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ ϱ *term-subst.is-ground-subst* γ
shows $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ ($\varrho \odot \gamma$)
proof−
have $\forall$ *x* $\in$ *range-vars'* ϱ. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (γ *x*) ($\mathcal{V}'$ *x*)
using *assms*
unfolding $welltyped_\sigma$*-def*
by *simp*

then have $\forall$ *x* $\in$ *range-vars'* ϱ. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (γ *x*) ($\mathcal{V}$ (*the-inv* ϱ (*Var x*)))
using *range-vars*
by *auto*

then have $\forall$ *x* $\in$ *range-vars'* ϱ. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (($\varrho \odot \gamma$) *x*) ($\mathcal{V}$ *x*)
by (*metis assms*(*1*) *eval-term.simps*(*1*) *subst-compose-def welltyped.Var well-typed*$_\sigma$*-welltyped*
welltyped-renaming)

then have $\forall$ *x* $\in$ *range-vars'* ϱ. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (*Var x* ·*t* ($\varrho \odot \gamma$)) ($\mathcal{V}$ *x*)
by *auto*

then have $\forall$ *x*. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (*Var x* ·*t* ($\varrho \odot \gamma$)) ($\mathcal{V}$ *x*)
by (*metis assms*(*1*) *eval-term.simps*(*1*) *subst-compose-def* $welltyped_\sigma$*-Var well-typed*$_\sigma$*-def*
$welltyped_\sigma$*-welltyped welltyped-renaming*)

then have $\forall$ *x* $\in$ *range-vars'* ϱ. *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (*Var x* ·*t* ϱ) ($\mathcal{V}$ *x*)
using $welltyped_\sigma$*-welltyped*[*OF assms*(*1*)]
by (*simp add*: *subst-compose-def*)

have $\forall x.$ *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (*Var* x $\cdot t$ ϱ) ($\mathcal{V}$ x)
by (*meson welltyped.Var welltyped-renaming*)

then have $\forall x.$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var* x $\cdot t$ ϱ) ($\mathcal{V}$ x)
using *welltyped-renaming*
by (*meson assms*(*2*) *welltyped*$_\sigma$*-welltyped*)

then show *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ ($\varrho \odot \gamma$)
unfolding *welltyped*$_\sigma$*-def*
by (*metis* (*mono-tags*, *lifting*) ‹$\forall x.$ *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (*Var* x $\cdot t$ $\varrho \odot \gamma$) ($\mathcal{V}$ x)› *assms*(*3*)
eval-term.simps(*1*) *term-subst.is-ground-subst-comp-right*
term-subst.is-ground-subst-is-ground term-subst.subst-ident-if-ground welltyped-renaming)
qed

lemma *welltyped*$_a$*-renaming*: *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ a $\longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}'$ (a $\cdot a$ ϱ)
using *welltyped-renaming*
unfolding *welltyped*$_a$*-def*
by(*cases a*)(*simp add*: *subst-atom*)

lemma *welltyped*$_l$*-renaming*: *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ l $\longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}'$ (l $\cdot l$ ϱ)
using *welltyped*$_a$*-renaming*
unfolding *welltyped*$_l$*-def*
by (*simp add*: *subst-literal*(*3*))

lemma *welltyped*$_c$*-renaming*: *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ c $\longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}'$ (c $\cdot$ ϱ)
using *welltyped*$_l$*-renaming*
unfolding *welltyped*$_c$*-def*
by (*simp add*: *clause.subst-def*)

end

context
fixes ϱ
assumes *renaming*: *term-subst.is-renaming* ϱ
begin

lemma *welltyped-renaming-weaker*:
assumes $\forall x \in$ *term.vars* (t $\cdot t$ ϱ). $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) = $\mathcal{V}'$ x
shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (t $\cdot t$ ϱ) τ
proof(*intro iffI*)
assume *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ
then show *welltyped* $\mathcal{F}$ $\mathcal{V}'$ (t $\cdot t$ ϱ) τ
using *assms*
proof(*induction rule*: *welltyped.induct*)
case (*Var* x τ)

obtain y **where** y: $Var\ x \cdot t\ \varrho = Var\ y$
  using *renaming*
  by (*metis eval-term.simps*(*1*) *term.collapse*(*1*) *term-subst-is-renaming-iff*)

then have $\mathcal{V}$ (*the-inv* ϱ (*Var y*)) $= \mathcal{V}'\ y$
  using *Var*(*2*)
  by *simp*

moreover have (*the-inv* ϱ (*Var y*)) $= x$
  using *y renaming*
  unfolding *term-subst-is-renaming-iff*
  by (*metis eval-term.simps*(*1*) *the-inv-f-f*)

ultimately have $\mathcal{V}'\ y = \tau$
  using *Var*
  by *argo*

then show *?case*
  unfolding *y*
  by(*rule welltyped.Var*)
next
  case (*Fun f* τs τ *ts*)

have *list-all2* (*welltyped* $\mathcal{F}$ $\mathcal{V}'$) (*map* ($\lambda s.\ s \cdot t\ \varrho$) *ts*) τs
  using *Fun*(*2*, *3*)
  by(*auto simp*: *list.rel-mono-strong list-all2-map1*)

then show *?case*
  by (*simp add*: *Fun.hyps welltyped.simps*)
qed
next
assume *welltyped* $\mathcal{F}$ $\mathcal{V}'$ ($t \cdot t\ \varrho$) τ
then show *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ
  using *assms*
proof(*induction t arbitrary*: τ)
  case (*Var x*)
  then obtain y **where** y: $Var\ x \cdot t\ \varrho = Var\ y$
    using *renaming*
    by (*metis eval-term.simps*(*1*) *term.collapse*(*1*) *term-subst-is-renaming-iff*)

then have $\mathcal{V}$ (*the-inv* ϱ (*Var y*)) $= \mathcal{V}'\ y$
  by (*simp add*: *Var*)

moreover have (*the-inv* ϱ (*Var y*)) $= x$
  using *y renaming*
  unfolding *term-subst-is-renaming-iff*
  by (*metis eval-term.simps*(*1*) *the-inv-f-f*)

moreover have $\mathcal{V}'\ y = \tau$
using *Var*
unfolding *y*
by (*meson right-uniqueD welltyped.Var welltyped-right-unique*)

ultimately have $\mathcal{V}\ x = \tau$
by *blast*

then show *?case*
by(*rule welltyped.Var*)
next
case (*Fun f ts*)
have $[\![\bigwedge x2a\ \tau.\ [\![x2a \in set\ ts;\ welltyped\ \mathcal{F}\ \mathcal{V}'\ (x2a\ \cdot t\ \varrho)\ \tau]\!] \Longrightarrow welltyped\ \mathcal{F}\ \mathcal{V}$ $x2a\ \tau;$
$welltyped\ \mathcal{F}\ \mathcal{V}'\ (Fun\ f\ (map\ (\lambda s.\ s\ \cdot t\ \varrho)\ ts))\ \tau;$
$\forall y{\in}set\ ts.\ \forall x{\in}term.vars\ (y\ \cdot t\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x]\!]$
$\Longrightarrow welltyped\ \mathcal{F}\ \mathcal{V}\ (Fun\ f\ ts)\ \tau$
by (*smt* (*verit*, *best*) *Term.term.simps*(*2*) *Term.term.simps*(*4*) *list.rel-mono-strong*

list-all2-map1 welltyped.simps)

with *Fun* **show** *?case*
by *auto*

qed
qed

lemma $welltyped_a$*-renaming-weaker*:
assumes $\forall x \in atom.vars\ (a\ \cdot a\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x$
shows $welltyped_a\ \mathcal{F}\ \mathcal{V}\ a \longleftrightarrow welltyped_a\ \mathcal{F}\ \mathcal{V}'\ (a\ \cdot a\ \varrho)$
proof(*cases a*)
case (*Upair a b*)

then have
$\bigwedge\tau.\ [\![\bigwedge t\ \mathcal{V}\ \mathcal{V}'\ \mathcal{F}\ \tau.$
$\forall x{\in}term.vars\ (t\ \cdot t\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x \Longrightarrow$
$welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau = welltyped\ \mathcal{F}\ \mathcal{V}'\ (t\ \cdot t\ \varrho)\ \tau;$
$\forall x{\in}term.vars\ (a\ \cdot t\ \varrho) \cup term.vars\ (b\ \cdot t\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x;$
$welltyped\ \mathcal{F}\ \mathcal{V}\ a\ \tau;$
$welltyped\ \mathcal{F}\ \mathcal{V}\ b\ \tau]\!]$
$\Longrightarrow \exists\tau.\ welltyped\ \mathcal{F}\ \mathcal{V}'\ (a\ \cdot t\ \varrho)\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}'\ (b\ \cdot t\ \varrho)\ \tau$
$\bigwedge\tau.\ [\![\ \bigwedge t\ \mathcal{V}\ \mathcal{V}'\ \mathcal{F}\ \tau.$
$\forall x{\in}term.vars\ (t\ \cdot t\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x \Longrightarrow$
$welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau = welltyped\ \mathcal{F}\ \mathcal{V}'\ (t\ \cdot t\ \varrho)\ \tau;$
$\forall x{\in}term.vars\ (a\ \cdot t\ \varrho) \cup term.vars\ (b\ \cdot t\ \varrho).\ \mathcal{V}\ (the\text{-}inv\ \varrho\ (Var\ x)) = \mathcal{V}'\ x;$
$welltyped\ \mathcal{F}\ \mathcal{V}'\ (a\ \cdot t\ \varrho)\ \tau;\ welltyped\ \mathcal{F}\ \mathcal{V}'\ (b\ \cdot t\ \varrho)\ \tau]\!]$
$\Longrightarrow \exists\tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ a\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}\ b\ \tau$
by (*metis UnCI welltyped-renaming-weaker*)+

with *Upair* **show** *?thesis*
using *welltyped-renaming-weaker assms*
unfolding $welltyped_a$*-def atom.vars-def*
by(*auto simp add*: *subst-atom*)
qed

lemma $welltyped_l$*-renaming-weaker*:
assumes $\forall x \in$ *literal.vars* $(l \cdot l\ \varrho)$. $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) $= \mathcal{V}'\ x$
shows $welltyped_l\ \mathcal{F}\ \mathcal{V}\ l \longleftrightarrow welltyped_l\ \mathcal{F}\ \mathcal{V}'\ (l \cdot l\ \varrho)$
using $welltyped_a$*-renaming-weaker assms*
unfolding $welltyped_l$*-def literal.vars-def set-literal-atm-of*
by (*simp add*: *subst-literal*(*3*))

lemma $welltyped_c$*-renaming-weaker*:
assumes $\forall x \in$ *clause.vars* $(c \cdot \varrho)$. $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) $= \mathcal{V}'\ x$
shows $welltyped_c\ \mathcal{F}\ \mathcal{V}\ c \longleftrightarrow welltyped_c\ \mathcal{F}\ \mathcal{V}'\ (c \cdot \varrho)$
using $welltyped_l$*-renaming-weaker assms*
unfolding $welltyped_c$*-def clause.vars-def clause.subst-def*
by *blast*

lemma *has-type-renaming-weaker*:
assumes $\forall x \in$ *term.vars* $(t \cdot t\ \varrho)$. $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) $= \mathcal{V}'\ x$
shows *has-type* $\mathcal{F}\ \mathcal{V}\ t\ \tau \longleftrightarrow$ *has-type* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
using *renaming assms*
proof(*cases t*)
case (*Var x1*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *Term.term.simps*(*4*) *assms eval-term.simps*(*1*) *has-type.simps is-Var-def*
renaming term.set-intros(*3*) *term-subst-is-renaming-iff the-inv-f-f*)
next
case (*Fun x21 x22*)
then show *?thesis*
by (*simp add*: *has-type.simps*)
qed

lemma $welltyped_\sigma$*-renaming-ground-subst-weaker*:
assumes
$welltyped_\sigma\ \mathcal{F}\ \mathcal{V}'\ \gamma$
$welltyped_\sigma$*-on* $X\ \mathcal{F}\ \mathcal{V}\ \varrho$
term-subst.is-ground-subst γ
$\forall x \in \bigcup$(*term.vars* ‘ ϱ ‘ X). $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) $= \mathcal{V}'\ x$
shows $welltyped_\sigma$*-on* $X\ \mathcal{F}\ \mathcal{V}\ (\varrho \odot \gamma)$
proof(*unfold* $welltyped_\sigma$*-on-def*, *intro ballI*)
fix x
assume $x \in X$

then have *welltyped* $\mathcal{F}\ \mathcal{V}\ (\varrho\ x)\ (\mathcal{V}\ x)$
using *assms*(*2*)

unfolding *welltyped*$_\sigma$*-on-def*
by *simp*

obtain *y* **where** *y*: ϱ *x* = *Var y*
by (*metis renaming term.collapse*(*1*) *term-subst-is-renaming-iff*)

then have $y \in \bigcup$(*term.vars* ' ϱ ' *X*)
using ‹*x* ∈ *X*›
by (*metis Union-iff image-eqI term.set-intros*(*3*))

moreover have *welltyped* $\mathcal{F}$ $\mathcal{V}$ (γ *y*) ($\mathcal{V}'$ *y*)
using *assms*(*1*)
by (*metis assms*(*3*) *emptyE eval-term.simps*(*1*) *term-subst.is-ground-subst-def*
welltyped$_\sigma$*-def*
welltyped-$\mathcal{V}$)

ultimately have *welltyped* $\mathcal{F}$ $\mathcal{V}$ (γ *y*) ($\mathcal{V}$ (*the-inv* ϱ (*Var y*)))
using *assms*(*4*)
by *metis*

moreover have *the-inv* ϱ (*Var y*) = *x*
using *y renaming*
by (*metis term-subst-is-renaming-iff the-inv-f-f*)

moreover have γ *y* = ($\varrho \odot \gamma$) *x*
using *y*
by (*simp add*: *subst-compose-def*)

ultimately show *welltyped* $\mathcal{F}$ $\mathcal{V}$ (($\varrho \odot \gamma$) *x*) ($\mathcal{V}$ *x*)
by *argo*
qed

end

lemma
infinite-even-nat: *infinite* { *n* :: *nat* . *even n* } **and**
infinite-odd-nat: *infinite* { *n* :: *nat* . *odd n* }
by (*metis Suc-leD dual-order.refl even-Suc infinite-nat-iff-unbounded-le mem-Collect-eq*)+

lemma *obtain-infinite-partition*:
obtains *X Y* :: *'a* :: {*countable*, *infinite*} *set*
where
X ∩ *Y* = {} *X* ∪ *Y* = *UNIV* **and**
infinite X **and**
infinite Y

proof−
  obtain *g* :: *'a* ⇒ *nat* **where** *bij g*
    using *countableE-infinite*[*of UNIV* :: *'a set*] *infinite-UNIV* **by** *blast*

  define *g'* **where** *g'* ≡ *inv g*

  then have *bij-g'*: *bij g'*
    by (*simp add*: ‹*bij g*› *bij-betw-inv-into*)

  define *X* :: *'a set* **where**
    X ≡ *g'* ‘ { *n*. *even n* }

  define *Y* :: *'a set* **where**
    Y ≡ *g'* ‘ { *n*. *odd n* }

  have *X* ∩ *Y* = {}
    using *bij-g'*
    unfolding *X-def Y-def*
    by (*simp add*: *bij-image-Collect-eq disjoint-iff*)

  moreover have *X* ∪ *Y* = *UNIV*
    using *bij-g'*
    unfolding *X-def Y-def*
    by(*auto simp*: *bij-image-Collect-eq*)

  moreover have *bij-betw g'* { *n*. *even n* } *X bij-betw g'* { *n*. *odd n* } *Y*
    unfolding *X-def Y-def*
    by (*metis* ‹*bij g*› *bij-betw-imp-surj-on g'-def inj-on-imp-bij-betw inj-on-inv-into top.extremum*)+

  then have *infinite X infinite Y*
    using *infinite-even-nat infinite-odd-nat bij-betw-finite*
    by *blast*+

  ultimately show *?thesis*
    using *that*
    by *blast*
qed

lemma ($\bigcup$*n'*.{ *n*. *g n* = *n'* }) = *UNIV*
  by *blast*

lemma *inv-enumerate*:
  assumes *infinite N*
  shows (λ*x*. *inv* (*enumerate N*) *x*) ‘ *N* = *UNIV*
  by (*metis assms enumerate-in-set inj-enumerate inv-f-eq surj-on-alternative*)

instance *nat* :: *infinite*
  by(*standard*) *simp*

lemma *finite-bij-enumerate-inv-into*:
  fixes S :: $'a$::*wellorder set*
  assumes S: *finite* S
  shows *bij-betw* (*inv-into* {..<*card* S} (*enumerate* S)) S {..<*card* S}
  using *finite-bij-enumerate*[*OF assms*] *bij-betw-inv-into*
  by *blast*

lemma *obtain-inj-test'-on*:
  fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: *nat* ⇒ *'ty*
  assumes
    finite X
    finite Y
    $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\ x = ty\}$
    $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty\}$
  obtains $f\ f'$ :: *nat* ⇒ *nat* **where**
    inj f *inj* f'
    f ‘ $X \cap f'$ ‘ $Y = \{\}$
    $\forall x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
    $\forall x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
proof
  have $\bigwedge ty.$ *infinite* $(\{x.\ \mathcal{V}_2\ x = ty\} - X)$
    by (*simp add*: *assms*(*1*) *assms*(*4*))

  then have *infinite*: $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty \wedge x \notin X\}$
    by (*simp add*: *set-diff-eq*)

  define f' **where**
    $\bigwedge x.\ f'\ x \equiv$ *enumerate* $\{y.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge y \notin X\}\ x$

  have *f'-not-in-x*: $\bigwedge x.\ f'\ x \notin X$
  proof−
    fix x
    show $f'\ x \notin X$
      unfolding *f'-def*
      using *enumerate-in-set*[*OF infinite*]
      by (*smt* (*verit*) *CollectD Collect-cong*)
  qed

   show *inj id*
    by *simp*

   show *inj* f'
   proof(*unfold inj-def*; *intro allI impI*)
    fix x y
    assume $f'\ x = f'\ y$

    moreover then have $\mathcal{V}_2\ y = \mathcal{V}_2\ x$

unfolding f'*-def*
by (*smt* (*verit*, *ccfv-SIG*) *Collect-mono-iff enumerate-in-set infinite mem-Collect-eq*

rev-finite-subset)

ultimately show $x = y$
unfolding f'*-def*
by (*smt* (*verit*) *Collect-cong infinite inj-enumerate inj-onD iso-tuple-UNIV-I*)
qed

show *id* ‘ $X \cap f'$ ‘ $Y = \{\}$
using f'*-not-in-x*
by *auto*

show $\forall x{\in}X.\ \mathcal{V}_1\ (id\ x) = \mathcal{V}_1\ x$
by *simp*

show $\forall x{\in}Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
unfolding f'*-def*
using *enumerate-in-set*[*OF infinite*]
by (*smt* (*verit*) *Collect-cong mem-Collect-eq*)
qed

lemma *obtain-inj''-on'*:
fixes $\mathcal{V}_1\ \mathcal{V}_2$ :: *'a* :: *infinite* ⇒ *'ty*
assumes *finite X finite Y* $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\ x = ty\}$ $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty\}$
obtains $f\ f'$ :: *'a* ⇒ *'a* **where**
inj f inj f'
f ‘ $X \cap f'$ ‘ $Y = \{\}$
$\forall x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
$\forall x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
proof
have $\bigwedge ty.$ *infinite* $(\{x.\ \mathcal{V}_2\ x = ty\} - X)$
by (*simp add*: *assms*(*1*) *assms*(*4*))

then have *infinite*: $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty \wedge x \notin X\}$
by (*simp add*: *set-diff-eq*)

have $\bigwedge ty.\ |\{x.\ \mathcal{V}_2\ x = ty\}| =o\ |\{x.\ \mathcal{V}_2\ x = ty\ \} - X|$
using *assms*(*1*, *4*)
using *card-of-infinite-diff-finite ordIso-symmetric* **by** *blast*

then have $\bigwedge ty.\ |\{x.\ \mathcal{V}_2\ x = ty\}| =o\ |\{x.\ \mathcal{V}_2\ x = ty \wedge x \notin X\}|$
using *set-diff-eq*[*of - X*]
by *auto*

then have *exists-g'*: $\bigwedge ty.\ \exists g'.$ *bij-betw* g' $\{x.\ \mathcal{V}_2\ x = ty\}$ $\{x.\ \mathcal{V}_2\ x = ty \wedge x \notin X\}$

using *card-of-ordIso* **by** *blast*

define *get-g′* **where**
$\bigwedge ty.\ get\text{-}g'\ ty \equiv SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = ty\}\ \{x.\ \mathcal{V}_2\ x = ty \land x \notin X\}$

define f' **where**
$\bigwedge x.\ f'\ x \equiv get\text{-}g'\ (\mathcal{V}_2\ x)\ x$

have *f′-not-in-x*: $\bigwedge x.\ f'\ x \notin X$
proof−
fix *y*

define g' **where** $g' \equiv SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\}$

have $y \in \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}$
by *simp*

moreover have $g'\ y \in \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\}$
proof−
have $\bigwedge g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\} \Longrightarrow$
$\mathcal{V}_2\ ((SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\})\ y) = \mathcal{V}_2\ y$
$\bigwedge g'.\ [\![bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\};$
$(SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \land x \notin X\})\ y \in X]\!]$
$\Longrightarrow False$
by (*smt* (*verit*, *ccfv-SIG*) *bij-betw-apply mem-Collect-eq verit-sko-ex-indirect*)+

then show *?thesis*
unfolding *g′-def*
using *exists-g′*[*of* $\mathcal{V}_2\ y$]
by *auto*
qed

then have $g'\ y \notin X$
by *simp*

then show $f'\ y \notin X$
unfolding *f′-def get-g′-def g′-def*.
qed

show *inj id*
by *simp*

show *inj* f'
proof(*unfold inj-def*; *intro allI impI*)
fix *x y*
assume $f'\ x = f'\ y$

moreover then have $\mathcal{V}_2\ y = \mathcal{V}_2\ x$
unfolding f'*-def get-*g'*-def*
using *someI-ex*[*OF exists-*g']
by (*smt* (*verit*, *best*) f'*-def get-*g'*-def bij-betw-iff-bijections calculation mem-Collect-eq*)

moreover have $\bigwedge g'. \ [\![(SOME\ g'.\ bij\text{-}betw\ g'\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x\}\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x \wedge xa \notin X\})\ x =$
$(SOME\ g'.\ bij\text{-}betw\ g'\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x\}\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x \wedge xa \notin X\})\ y;$
$\mathcal{V}_2\ y = \mathcal{V}_2\ x;\ \bigwedge P\ x.\ P\ x \Longrightarrow P\ (Eps\ P);$
$bij\text{-}betw\ g'\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x\}\ \{xa.\ \mathcal{V}_2\ xa = \mathcal{V}_2\ x \wedge xa \notin X\}]\!]$
$\Longrightarrow x = y$
by (*smt* (*verit*, *ccfv-threshold*) *bij-betw-iff-bijections mem-Collect-eq some-eq-ex*)

ultimately show $x = y$
using *exists-*g'[*of* $\mathcal{V}_2\ x$] *someI*
unfolding f'*-def get-*g'*-def*
by *auto*
qed

show $id\ `\ X \cap f'\ `\ Y = \{\}$
using f'*-not-in-x*
by *auto*

show $\forall x \in X.\ \mathcal{V}_1\ (id\ x) = \mathcal{V}_1\ x$
by *simp*

show $\forall y \in Y.\ \mathcal{V}_2\ (f'\ y) = \mathcal{V}_2\ y$
proof(*intro ballI*)
fix y
assume $y \in Y$

define g' **where** $g' \equiv SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\}$

have $y \in \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}$
by *simp*

have $g'\ y \in \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\}$
proof−
have $\bigwedge g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\} \Longrightarrow$
$\mathcal{V}_2\ ((SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\})\ y) = \mathcal{V}_2\ y$
$\bigwedge g'.\ [\![bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\};$
$(SOME\ g'.\ bij\text{-}betw\ g'\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y\}\ \{x.\ \mathcal{V}_2\ x = \mathcal{V}_2\ y \wedge x \notin X\})\ y \in X]\!]$
$\Longrightarrow False$
by (*smt* (*verit*, *ccfv-SIG*) *bij-betw-apply mem-Collect-eq verit-sko-ex-indirect*)+

then show *?thesis*
unfolding *g′-def*
using *exists-g′*[*of* $\mathcal{V}_2$ *y*]
by *auto*
qed

then show $\mathcal{V}_2$ (*f′ y*) = $\mathcal{V}_2$ *y*
unfolding *g′-def f′-def get-g′-def*
by *blast*
qed
qed

lemma *obtain-inj′′-on*:
fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: *′a* :: {*countable*, *infinite*} ⇒ *′ty*
assumes *finite X finite Y* ⋀*ty*. *infinite* {*x*. $\mathcal{V}_1$ *x* = *ty*} ⋀*ty*. *infinite* {*x*. $\mathcal{V}_2$ *x* = *ty*}
obtains *f f′* :: *′a* ⇒ *′a* **where**
inj f inj f′
f ‘ *X* ∩ *f′* ‘ *Y* = {}
∀ *x* ∈ *X*. $\mathcal{V}_1$ (*f x*) = $\mathcal{V}_1$ *x*
∀ *x* ∈ *Y*. $\mathcal{V}_2$ (*f′ x*) = $\mathcal{V}_2$ *x*
proof−
obtain *a-to-nat* :: *′a* ⇒ *nat* **where** *bij-a-to-nat*: *bij a-to-nat*
using *countableE-infinite*[*of UNIV* :: *′a set*] *infinite-UNIV* **by** *blast*

define *nat-to-a* **where** *nat-to-a* ≡ *inv a-to-nat*

have *bij-nat-to-a*: *bij nat-to-a*
unfolding *nat-to-a-def*
by (*simp add*: *bij-a-to-nat bij-imp-bij-inv*)

define *X-nat Y-nat* **where**
X-nat ≡ *a-to-nat* ‘ *X* **and**
Y-nat ≡ *a-to-nat* ‘ *Y*

have *finite-X-nat*: *finite X-nat* **and** *finite-Y-nat*: *finite Y-nat*
unfolding *X-nat-def Y-nat-def*
using *assms*(*1*,*2*)
by *blast*+

define $\mathcal{V}_1$*-nat* $\mathcal{V}_2$*-nat* **where**
⋀*n*. $\mathcal{V}_1$*-nat n* ≡ $\mathcal{V}_1$ (*nat-to-a n*) **and**
⋀*n*. $\mathcal{V}_2$*-nat n* ≡ $\mathcal{V}_2$ (*nat-to-a n*)

have
⋀*ty*. {*x*. $\mathcal{V}_1$*-nat x* = *ty*} = *a-to-nat* ‘ {*x*. $\mathcal{V}_1$ *x* = *ty*}
⋀*ty*. {*x*. $\mathcal{V}_2$*-nat x* = *ty*} = *a-to-nat* ‘ {*x*. $\mathcal{V}_2$ *x* = *ty*}

unfolding $\mathcal{V}_1$*-nat-def* $\mathcal{V}_2$*-nat-def*
using *bij-a-to-nat bij-image-Collect-eq nat-to-a-def* **by** *fastforce+*

then have $\mathcal{V}$*-nat-infinite*: $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\text{-}nat\ x = ty\}$ $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\text{-}nat\ x = ty\}$
using *assms*(*3*, *4*)
by (*metis bij-a-to-nat bij-betw-finite bij-betw-subset subset-UNIV*)+

obtain *f-nat f′-nat* **where**
inj: *inj f-nat inj f′-nat* **and**
disjoint: *f-nat ‘ X-nat* ∩ *f′-nat ‘ Y-nat* = {} **and**
type-preserving:
∀ *x*∈ *X-nat*. $\mathcal{V}_1$*-nat* (*f-nat x*) = $\mathcal{V}_1$*-nat x*
∀ *x*∈ *Y-nat*. $\mathcal{V}_2$*-nat* (*f′-nat x*) = $\mathcal{V}_2$*-nat x*
using *obtain-inj-test′-on*[*OF finite-X-nat finite-Y-nat* $\mathcal{V}$*-nat-infinite*]**.**

let *?f* = *nat-to-a* ∘ *f-nat* ∘ *a-to-nat*
let *?f′* = *nat-to-a* ∘ *f′-nat* ∘ *a-to-nat*

have *inj ?f inj ?f′*
using *inj*
by (*simp-all add*: *bij-a-to-nat bij-is-inj bij-nat-to-a inj-compose*)

moreover have *?f ‘ X* ∩ *?f′ ‘ Y* = {}
using *disjoint*
unfolding *X-nat-def Y-nat-def*
by (*metis bij-is-inj bij-nat-to-a image-Int image-comp image-empty*)

moreover have
∀ *x*∈ *X*. $\mathcal{V}_1$ (*?f x*) = $\mathcal{V}_1$ *x*
∀ *x*∈ *Y*. $\mathcal{V}_2$ (*?f′ x*) = $\mathcal{V}_2$ *x*
using *type-preserving*
unfolding *X-nat-def Y-nat-def* $\mathcal{V}_1$*-nat-def* $\mathcal{V}_2$*-nat-def*
by (*simp-all add*: *bij-a-to-nat bij-is-inj nat-to-a-def*)

ultimately show *?thesis*
using *that*
by *presburger*
qed

lemma *obtain-inj′*:
obtains *f* :: *′a* :: *infinite* ⇒ *′a* **where**
inj f
|*range f*| =*o* |*UNIV* − *range f*|
proof−
obtain *X Y* :: *′a set* **where**
X-Y:
|*X*| =*o* |*Y*|

$|X| =o |UNIV :: 'a\ set|$
$X \cap Y = \{\}$
$X \cup Y = UNIV$
using *partitions*[*OF infinite-UNIV*]
by *blast*

then obtain *f* **where**
f: *bij-betw f* (*UNIV* :: *'a set*) *Y*
by (*meson card-of-ordIso ordIso-symmetric ordIso-transitive*)

have *inj-f*: *inj f*
using *f bij-betw-def* **by** *blast*+

have *Y*: *Y* = *range f*
using *f*
by (*simp add*: *bij-betw-def*)

have *X*: *X* = *UNIV* − *range f*
using *X-Y*
unfolding *Y*
by *auto*

show *?thesis*
using *X X-Y*(*1*) *Y inj-f ordIso-symmetric that* **by** *blast*
qed

lemma *obtain-inj*:
fixes *X*
defines $Y \equiv UNIV - X$
assumes
infinite-X: *infinite X* **and**
infinite-Y: *infinite Y*
obtains $f :: 'a :: \{countable, infinite\} \Rightarrow 'a$ **where**
inj f
$range\ f \cap X = \{\}$
$range\ f \cup X = UNIV$
proof−
obtain $g :: 'a \Rightarrow nat$ **where** *bij*: *bij g*
using *countableE-infinite*[*of UNIV* :: *'a set*] *infinite-UNIV* **by** *blast*

have *X-Y*: $X \cap Y = \{\}$ $X \cup Y = UNIV$
unfolding *Y-def*
by *simp-all*

have *countable-X*: *countable X* **and** *countable-Y*: *countable Y*
by *auto*

obtain *f* **where**
f: *bij-betw f* (*UNIV* :: *'a set*) *Y*

using *countable-infiniteE′*[*OF countable-Y infinite-Y*]
by (*meson bij bij-betw-trans*)

have *inj f*
using *f bij-betw-def* **by** *blast+*

moreover have *range f = Y*
using *f*
by (*simp-all add*: *bij-betw-def*)

then have *range f ∩ X = {} range f ∪ X = UNIV*
using *X-Y*
by *auto*

ultimately show *?thesis*
using *that*
by *presburger*
qed

lemma *obtain-injs*:
obtains *f f′* :: *′a* :: {*countable*, *infinite*} ⇒ *′a* **where**
inj f inj f′
range f ∩ range f′ = {}
range f ∪ range f′ = UNIV
proof−
obtain *g* :: *′a ⇒ nat* **where** *bij g*
using *countableE-infinite*[*of UNIV* :: *′a set*] *infinite-UNIV* **by** *blast*

define *g′* **where** *g′ ≡ inv g*

then have *bij-g′*: *bij g′*
by (*simp add*: ‹*bij g*› *bij-betw-inv-into*)

obtain *X Y* :: *′a set* **where**
X-Y: *X ∩ Y = {} X ∪ Y = UNIV* **and**
infinite-X: *infinite X* **and**
infinite-Y: *infinite Y*
using *obtain-infinite-partition*
by *auto*

have *countable-X*: *countable X* **and** *countable-Y*: *countable Y*
by *blast+*

obtain *f* **where**
f: *bij-betw f* (*UNIV* :: *′a set*) *X*
using *countable-infiniteE′*[*OF countable-X infinite-X*]
by (*meson* ‹*bij g*› *bij-betw-trans*)

obtain *f′* **where**

f': *bij-betw* f' (*UNIV* :: *'a set*) Y
using *countable-infiniteE'*[*OF countable-Y infinite-Y*]
by (*meson* ‹*bij g*› *bij-betw-trans*)

have *inj f inj f'*
using *f f' bij-betw-def* **by** *blast*+

moreover have *range f* = *X range f'* = *Y*
using *f f'*
by (*simp-all add*: *bij-betw-def*)

then have *range f* ∩ *range f'* = {} *range f* ∪ *range f'* = *UNIV*
using *X-Y*
by *simp-all*

ultimately show *?thesis*
using *that*
by *presburger*
qed

lemma *welltyped-on-renaming-exists'*:
assumes *finite X finite Y* ⋀*ty. infinite* {*x.* $\mathcal{V}_1$ *x* = *ty*} ⋀*ty. infinite* {*x.* $\mathcal{V}_2$ *x* = *ty*}
obtains ϱ_1 ϱ_2 :: (*'f*, *'v* :: *infinite*) *subst* **where**
term-subst.is-renaming ϱ_1
term-subst.is-renaming ϱ_2
ϱ_1 ‘ *X* ∩ ϱ_2 ‘ *Y* = {}
welltyped$_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}_1$ ϱ_1
welltyped$_\sigma$*-on Y* $\mathcal{F}$ $\mathcal{V}_2$ ϱ_2
proof−
obtain *renaming*$_1$ *renaming*$_2$ :: *'v* ⇒ *'v* **where**
renamings:
inj renaming$_1$ *inj renaming*$_2$
renaming$_1$ ‘ *X* ∩ *renaming*$_2$ ‘ *Y* = {}
∀ *x* ∈ *X*. $\mathcal{V}_1$ (*renaming*$_1$ *x*) = $\mathcal{V}_1$ *x*
∀ *x* ∈ *Y*. $\mathcal{V}_2$ (*renaming*$_2$ *x*) = $\mathcal{V}_2$ *x*
using *obtain-inj''-on'*[*OF assms*]**.**

define ϱ_1 :: (*'f*, *'v*) *subst* **where**
⋀*x*. ϱ_1 *x* ≡ *Var* (*renaming*$_1$ *x*)

define ϱ_2 :: (*'f*, *'v*) *subst* **where**
⋀*x*. ϱ_2 *x* ≡ *Var* (*renaming*$_2$ *x*)

have *term-subst.is-renaming* ϱ_1 *term-subst.is-renaming* ϱ_2
unfolding ϱ_1*-def* ϱ_2*-def*
using *renamings*(*1*,*2*)
by (*meson injD injI term-subst.is-renaming-id-subst term-subst-is-renaming-iff*)+

moreover have ϱ_1 ‘ $X \cap \varrho_2$ ‘ $Y = \{\}$
unfolding ϱ_1-*def* ϱ_2-*def* *range-vars′-def*
using *renamings*(*3*)
by *auto*

moreover have *welltyped*$_\sigma$-*on* X $\mathcal{F}$ $\mathcal{V}_1$ ϱ_1 *welltyped*$_\sigma$-*on* Y $\mathcal{F}$ $\mathcal{V}_2$ ϱ_2
unfolding ϱ_1-*def* ϱ_2-*def* *welltyped*$_\sigma$-*on-def*
using *renamings*(*4*, *5*)
by(*auto simp*: *welltyped.Var*)

ultimately show *?thesis*
using *that*
by *presburger*
qed

lemma *welltyped-on-renaming-exists*:
assumes *finite X finite Y* $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\ x = ty\}$ $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty\}$
obtains ϱ_1 ϱ_2 :: (*′f*, *′v* :: {*countable*, *infinite*}) *subst* **where**
term-subst.is-renaming ϱ_1
term-subst.is-renaming ϱ_2
ϱ_1 ‘ $X \cap \varrho_2$ ‘ $Y = \{\}$
welltyped$_\sigma$-*on* X $\mathcal{F}$ $\mathcal{V}_1$ ϱ_1
welltyped$_\sigma$-*on* Y $\mathcal{F}$ $\mathcal{V}_2$ ϱ_2
proof−
obtain *renaming*$_1$ *renaming*$_2$ :: *′v* ⇒ *′v* **where**
renamings:
inj renaming$_1$ *inj renaming*$_2$
renaming$_1$ ‘ $X \cap$ *renaming*$_2$ ‘ $Y = \{\}$
$\forall x \in X.\ \mathcal{V}_1\ (renaming_1\ x) = \mathcal{V}_1\ x$
$\forall x \in Y.\ \mathcal{V}_2\ (renaming_2\ x) = \mathcal{V}_2\ x$
using *obtain-inj′′-on*[*OF assms*]**.**

define ϱ_1 :: (*′f*, *′v*) *subst* **where**
$\bigwedge x.\ \varrho_1\ x \equiv Var\ (renaming_1\ x)$

define ϱ_2 :: (*′f*, *′v*) *subst* **where**
$\bigwedge x.\ \varrho_2\ x \equiv Var\ (renaming_2\ x)$

have *term-subst.is-renaming* ϱ_1 *term-subst.is-renaming* ϱ_2
unfolding ϱ_1-*def* ϱ_2-*def*
using *renamings*(*1*,*2*)
by (*meson injD injI term-subst.is-renaming-id-subst term-subst-is-renaming-iff*)+

moreover have ϱ_1 ‘ $X \cap \varrho_2$ ‘ $Y = \{\}$
unfolding ϱ_1-*def* ϱ_2-*def* *range-vars′-def*
using *renamings*(*3*)
by *auto*

moreover have *welltyped*$_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}_1$ ϱ_1 *welltyped*$_\sigma$*-on Y* $\mathcal{F}$ $\mathcal{V}_2$ ϱ_2
unfolding ϱ_1*-def* ϱ_2*-def welltyped*$_\sigma$*-on-def*
using *renamings(4, 5)*
by(*auto simp*: *welltyped.Var*)

ultimately show *?thesis*
using *that*
by *presburger*
qed

lemma *welltyped*$_\sigma$*-subst-upd*:
assumes *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var var*) τ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *update* τ *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ γ
shows *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (γ(*var* := *update*))
using *assms*
unfolding *welltyped*$_\sigma$*-def*
by (*metis fun-upd-other fun-upd-same right-unique-def welltyped.Var welltyped-right-unique*)

lemma *welltyped*$_\sigma$*-on-subst-upd*:
assumes *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var var*) τ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *update* τ *welltyped*$_\sigma$*-on X* $\mathcal{F}$
$\mathcal{V}$ γ
shows *welltyped*$_\sigma$*-on X* $\mathcal{F}$ $\mathcal{V}$ (γ(*var* := *update*))
using *assms*
unfolding *welltyped*$_\sigma$*-on-def*
by (*metis fun-upd-other fun-upd-same right-unique-def welltyped.Var welltyped-right-unique*)

lemma *welltyped-is-ground*:
assumes *term.is-ground t welltyped* $\mathcal{F}$ $\mathcal{V}$ *t* τ
shows *welltyped* $\mathcal{F}$ $\mathcal{V}'$ *t* τ
by (*metis assms(1) assms(2) empty-iff welltyped-*$\mathcal{V}$)

lemma *term-subst-is-imgu-is-mgu*: *term-subst.is-imgu* μ {{*s*, *t*}} = *is-imgu* μ {(*s*,
t)}
apply (*simp add*: *term-subst-is-imgu-iff-is-imgu*)
by (*smt* (*verit*, *ccfv-threshold*) *insert-absorb2 insert-commute is-imgu-def unifiers-insert-ident*
unifiers-insert-swap)

lemma *the-mgu-term-subst-is-imgu*:
fixes σ :: (*'f*, *'v*) *subst*
assumes *s* ·*t* σ = *t* ·*t* σ
shows *term-subst.is-imgu* (*the-mgu s t*) {{*s*, *t*}}
using *term-subst-is-imgu-is-mgu the-mgu-is-imgu*
using *assms* **by** *blast*

lemma *Fun-arg-types*:
assumes
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun f fs*) τ
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun f gs*) τ
obtains τs **where**

$\mathcal{F}\ f = (\tau s,\ \tau)$
list-all2 (*welltyped* $\mathcal{F}\ \mathcal{V}$) *fs* τs
list-all2 (*welltyped* $\mathcal{F}\ \mathcal{V}$) *gs* τs
by (*smt* (*verit*, *ccfv-SIG*) *Pair-inject assms*(*1*) *assms*(*2*) *option.inject term.distinct*(*1*) *term.inject*(*2*) *welltyped.simps*)

lemma *welltyped-zip-option*:
assumes
welltyped $\mathcal{F}\ \mathcal{V}$ (*Fun f ts*) τ
welltyped $\mathcal{F}\ \mathcal{V}$ (*Fun f ss*) τ
zip-option ts ss = *Some ds*
shows
$\forall (a,\ b) \in$ *set ds*. $\exists \tau$. *welltyped* $\mathcal{F}\ \mathcal{V}$ *a* τ $\wedge$ *welltyped* $\mathcal{F}\ \mathcal{V}$ *b* τ
proof−

obtain τs **where**
list-all2 (*welltyped* $\mathcal{F}\ \mathcal{V}$) *ts* τs
list-all2 (*welltyped* $\mathcal{F}\ \mathcal{V}$) *ss* τs
using *Fun-arg-types*[*OF assms*(*1*, *2*)].

with *assms*(*3*) **show** *?thesis*
proof(*induction ts ss arbitrary*: τs *ds rule*: *zip-induct*)
case (*Cons-Cons t ts s ss*)
then obtain $\tau'\ \tau s'$ **where** τs: $\tau s = \tau' \# \tau s'$
by (*meson list-all2-Cons1*)

from *Cons-Cons*(*2*)
obtain $d'\ ds'$ **where** *ds*: $ds = d' \# ds'$
by *auto*

have *zip-option ts ss* = *Some* ds'
using *Cons-Cons*(*2*)
unfolding *ds*
by *fastforce*

moreover have *list-all2* (*welltyped* $\mathcal{F}\ \mathcal{V}$) *ts* $\tau s'$
using *Cons-Cons.prems*(*2*) τs **by** *blast*

moreover have *list-all2* (*welltyped* $\mathcal{F}\ \mathcal{V}$) *ss* $\tau s'$
using *Cons-Cons.prems*(*3*) τs **by** *blast*

ultimately have $\forall (t,\ s) \in$ *set* ds'. $\exists \tau$. *welltyped* $\mathcal{F}\ \mathcal{V}$ *t* τ $\wedge$ *welltyped* $\mathcal{F}\ \mathcal{V}$ *s* τ
using *Cons-Cons.IH*
by *presburger*

moreover have $\exists \tau$. *welltyped* $\mathcal{F}\ \mathcal{V}$ *t* τ $\wedge$ *welltyped* $\mathcal{F}\ \mathcal{V}$ *s* τ
using *Cons-Cons.prems*(*2*) *Cons-Cons.prems*(*3*) τs **by** *blast*

ultimately show *?case*

using *Cons-Cons.prems*(*1*) *ds*
by *fastforce*
qed(*auto*)
qed

lemma *welltyped-decompose'*:
assumes
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun f fs*) τ
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun f gs*) τ
decompose (*Fun f fs*) (*Fun g gs*) = *Some ds*
shows $\forall (t, t') \in$ *set ds*. $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
using *assms welltyped-zip-option*[*OF assms*(*1*,*2*)]
by *force*

lemma *welltyped-decompose*:
assumes
welltyped $\mathcal{F}$ $\mathcal{V}$ *f* τ
welltyped $\mathcal{F}$ $\mathcal{V}$ *g* τ
decompose f g = *Some ds*
shows $\forall (t, t') \in$ *set ds*. $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
proof−

obtain *f' fs gs* **where** *f* = *Fun f' fs g* = *Fun f' gs*
using *assms*(*3*)
unfolding *decompose-def*
by (*smt* (*z3*) *option.distinct*(*1*) *prod.simps*(*2*) *rel-option-None1 term.split-sels*(*2*))

then show *?thesis*
using *assms welltyped-decompose'*
by (*metis* (*mono-tags*, *lifting*))
qed

lemma *welltyped-subst'-subst*:
assumes *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var x*) τ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *t* τ
shows $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst x t*)
using *assms*
unfolding *subst-def* $welltyped_\sigma$*-def*
by (*simp add*: *welltyped.simps*)

lemma *welltyped-unify*:
assumes
unify es bs = *Some unifier*
$\forall (t, t') \in$ *set es*. $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
$welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst-of bs*)
shows $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst-of unifier*)
using *assms*
proof(*induction es bs arbitrary*: *unifier rule*: *unify.induct*)
case (*1 bs*)
then show *?case*

by *simp*
next
case (*2 f ss g ts E bs*)
then obtain τ **where** τ:
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun f ss*) τ
welltyped $\mathcal{F}$ $\mathcal{V}$ (*Fun g ts*) τ
by *auto*

obtain *ds* **where** *ds*: *decompose* (*Fun f ss*) (*Fun g ts*) = *Some ds*
using *2*(*2*)
by(*simp split*: *option.splits*)

moreover then have *unify* (*ds* @ *E*) *bs* = *Some unifier*
using *2.prems*(*1*) **by** *auto*

moreover have $\forall (t, t')\in$*set* (*ds* @ *E*). $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ *t* τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
using *welltyped-decompose*[*OF* τ *ds*] *2*(*3*)
by *fastforce*

ultimately show *?case*
using *2*
by *blast*
next
case (*3 x t E bs*)
show *?case*
proof(*cases t* = *Var x*)
case *True*
then show *?thesis*
using *3*
by *simp*
next
case *False*
then have *unify* (*subst-list* (*subst x t*) *E*) ((*x*, *t*) # *bs*) = *Some unifier*
using *3*
by(*auto split*: *if-splits*)

moreover have
$\forall (s, s') \in$ *set E*. $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*s* $\cdot$*t Var*(*x* := *t*)) τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (s' $\cdot$*t Var*(*x* := *t*)) τ
using *3*(*4*)
by (*smt* (*verit*, *ccfv-threshold*) *case-prodD case-prodI2 fun-upd-apply welltyped.Var*
list.set-intros(*1*) *list.set-intros*(*2*) *right-uniqueD welltyped-right-unique*
welltyped$_\sigma$*-def welltyped*$_\sigma$*-welltyped*)

moreover then have
$\forall (s, s') \in$ *set* (*subst-list* (*subst x t*) *E*). $\exists\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ *s* τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ s' τ

unfolding *subst-def subst-list-def*
by *fastforce*

moreover have $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst x t*)
using *3*(*4*) *welltyped-subst′-subst*
by *fastforce*

moreover then have $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst-of* ((*x*, *t*) # *bs*))
using *3*(*5*)
unfolding $welltyped_\sigma$*-def*
by (*simp add*: *calculation*(*4*) *subst-compose-def* $welltyped_\sigma$*-welltyped*)

ultimately show *?thesis*
using *3*(*2*, *3*) *False* **by** *force*
qed
next
case (*4 t ts x E bs*)
then have *unify* (*subst-list* (*subst x* (*Fun t ts*)) *E*) ((*x*, (*Fun t ts*)) # *bs*) = *Some unifier*
by(*auto split*: *if-splits*)

moreover have
$\forall (s, s') \in$ *set E*. $\exists \tau$.
welltyped $\mathcal{F}$ $\mathcal{V}$ ($s \cdot t$ *Var*($x :=$ (*Fun t ts*))) $\tau \wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ ($s' \cdot t$ *Var*($x :=$ (*Fun t ts*))) τ
using *4*(*3*)
by (*smt* (*verit*, *ccfv-threshold*) *case-prodD case-prodI2 fun-upd-apply welltyped.Var*

list.set-intros(*1*) *list.set-intros*(*2*) *right-uniqueD welltyped-right-unique*
$welltyped_\sigma$*-def* $welltyped_\sigma$*-welltyped*)

moreover then have
$\forall (s, s') \in$ *set* (*subst-list* (*subst x* (*Fun t ts*)) *E*). $\exists \tau$.
welltyped $\mathcal{F}$ $\mathcal{V}$ *s* $\tau \wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ s' τ
unfolding *subst-def subst-list-def*
by *fastforce*

moreover have $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst x* (*Fun t ts*))
using *4*(*3*) *welltyped-subst′-subst*
by *fastforce*

moreover then have $welltyped_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst-of* ((*x*, (*Fun t ts*)) # *bs*))
using *4*(*4*)
unfolding $welltyped_\sigma$*-def*
by (*simp add*: *calculation*(*4*) *subst-compose-def* $welltyped_\sigma$*-welltyped*)

ultimately show *?case*
using *4*(*1*, *2*)
by (*metis* (*no-types*, *lifting*) *option.distinct*(*1*) *unify.simps*(*4*))

qed

lemma *welltyped-unify'*:
  assumes
    unify: *unify* $[(t,\ t')]$ [] = *Some unifier* **and**
    τ: $\exists\,\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
  shows *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ (*subst-of unifier*)
  using *assms welltyped-unify*[*OF unify*] τ *welltyped*$_\sigma$*-Var*
  by *fastforce*

lemma *welltyped-the-mgu*:
  assumes
    the-mgu: *the-mgu* t t' = μ **and**
    τ: $\exists\,\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ
  shows
    welltyped$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ μ
  using *assms welltyped-unify'*[*of* t t' - $\mathcal{F}$ $\mathcal{V}$]
  unfolding *the-mgu-def mgu-def welltyped*$_\sigma$*-def*
  by(*auto simp*: *welltyped.Var split*: *option.splits*)

abbreviation *welltyped-imgu* **where**
  welltyped-imgu $\mathcal{F}$ $\mathcal{V}$ *term term'* μ $\equiv$
    $\forall\,\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term* τ $\longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term'* τ $\longrightarrow$ *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ μ

lemma *welltyped-imgu-exists*:
  fixes υ :: ('*f*, '*v*) *subst*
  assumes *unified*: *term* $\cdot t$ υ = *term'* $\cdot t$ υ
  obtains μ :: ('*f*, '*v*) *subst*
  where
    $\upsilon = \mu \odot \upsilon$
    term-subst.is-imgu μ {{*term*, *term'*}}
    welltyped-imgu $\mathcal{F}$ $\mathcal{V}$ *term term'* μ
proof−
  obtain μ **where** μ: *the-mgu term term'* = μ
    using *assms ex-mgu-if-subst-apply-term-eq-subst-apply-term* **by** *blast*

  have *welltyped-imgu* $\mathcal{F}$ $\mathcal{V}$ *term term'* (*the-mgu term term'*)
    using *welltyped-the-mgu*[*OF* μ, *of* $\mathcal{F}$ $\mathcal{V}$] *assms*
    unfolding μ
    by *blast*

  then show *?thesis*
    using *that imgu-exists-extendable*[*OF unified*]
    by (*metis the-mgu the-mgu-term-subst-is-imgu unified*)
qed

abbreviation *welltyped-imgu'* **where**
  welltyped-imgu' $\mathcal{F}$ $\mathcal{V}$ *term term'* μ $\equiv$
    $\exists\,\tau$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term* τ $\wedge$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term'* τ $\wedge$ *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ μ

lemma *welltyped-imgu′-exists*:
  fixes υ :: ($'f$, $'v$) *subst*
  assumes *unified*: *term* $\cdot t$ υ = *term′* $\cdot t$ υ **and** *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term* τ *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term′* τ
  obtains μ :: ($'f$, $'v$) *subst*
  where
    $\upsilon = \mu \odot \upsilon$
    term-subst.is-imgu μ {{*term*, *term′*}}
    welltyped-imgu′ $\mathcal{F}$ $\mathcal{V}$ *term term′* μ
proof−
  obtain μ **where** μ: *the-mgu term term′* = μ
    using *assms ex-mgu-if-subst-apply-term-eq-subst-apply-term* **by** *blast*

  have *welltyped-imgu* $\mathcal{F}$ $\mathcal{V}$ *term term′* (*the-mgu term term′*)
    using *welltyped-the-mgu*[*OF* μ, *of* $\mathcal{F}$ $\mathcal{V}$] *assms*
    unfolding μ
    by *blast*

  then show *?thesis*
    using *that imgu-exists-extendable*[*OF unified*]
    by (*metis assms*(*2*) *assms*(*3*) *the-mgu the-mgu-term-subst-is-imgu unified*)
qed

end
theory *First-Order-Select*
  imports
    Selection-Function
    First-Order-Clause
    First-Order-Type-System
begin

type-synonym ($'f$, $'v$, $'ty$) *typed-clause* = ($'f$, $'v$) *atom clause* × ($'v$, $'ty$) *var-types*

type-synonym $'f$ *ground-select* = $'f$ *ground-atom clause* ⇒ $'f$ *ground-atom clause*
type-synonym ($'f$, $'v$) *select* = ($'f$, $'v$) *atom clause* ⇒ ($'f$, $'v$) *atom clause*

definition *is-select-grounding* :: ($'f$, $'v$) *select* ⇒ $'f$ *ground-select* ⇒ *bool* **where**
  ⋀*select* $select_G$.
      is-select-grounding select $select_G$ = (∀ $clause_G$. ∃ *clause* γ.
      clause.is-ground (*clause* · γ) ∧
      $clause_G$ = *clause.to-ground* (*clause* · γ) ∧
      $select_G$ $clause_G$ = *clause.to-ground* ((*select clause*) · γ))

lemma *infinite-lists-per-length*: *infinite* {*l* :: ($'a$ :: *infinite*) *list*. *length* (*tl l*) = *y*}
proof(*induction y*)
  case *0*

  show *?case*

proof
  assume *a*: *finite* {*l* :: *'a list*. *length* (*tl l*) = *0*}

  define *f* **where** $\bigwedge x$:: *'a* . *f x* ≡ [*x*]

  have $\bigwedge x$ *y*. *f x* = *f y* ⟹ *x* = *y*
    unfolding *f-def*
    by (*metis nth-Cons-0*)

  moreover have $\bigwedge x$. *length* (*f x*) ≤ *Suc 0*
    unfolding *f-def*
    by *simp*

  moreover have $\bigwedge x$. *length x* = *Suc 0* ⟹ *x* ∈ *range f*
    unfolding *f-def*
    by (*smt* (*z3*) *One-nat-def Suc-length-conv Suc-pred' diff-Suc-1 diff-is-0-eq'*
      length-0-conv nat.simps(*3*) *not-gr0 rangeI*)

  moreover have $\bigwedge x$. ⟦*x* ∉ *range f*; *length x* ≤ *Suc 0*⟧ ⟹ *x* = []
    using *calculation*(*3*) *le-Suc-eq* **by** *auto*

  moreover have $\bigwedge$*xa*. *f xa* = [] ⟹ *False*
    unfolding *f-def*
    by *simp*

  ultimately have *tt*: *bij-betw f UNIV* ({*l*. *length* (*tl l*) = *0*} − {[]})
    unfolding *bij-betw-def inj-def*
    by *auto presburger*

  then have *infinite* ({*l* :: *'a list*. *length* (*tl l*) = *0*} − {[]})
    using *bij-betw-finite infinite-UNIV* **by** *blast*

  then have *infinite* {*l* :: *'a list*. *length* (*tl l*) = *0*}
    by *simp*

  with *a* **show** *False*
    by *blast*
qed
next
case (*Suc y*)

have *1*: {*l* :: *'a list*. *length* (*tl l*) = *y*} =
  (*if y* = *0 then insert* [] {*l*. *length l* = *1*} *else* {*l*. *length l* = *Suc y*})
  by (*auto simp*: *le-Suc-eq*)

have *2*: $\bigwedge x$. *length x* = *Suc y* ⟹ *x* ∈ *tl* ‘ {*l*. *length l* − *Suc 0* = *Suc y*}
by (*metis* (*mono-tags*, *lifting*) *One-nat-def imageI length-tl list.sel*(*3*) *mem-Collect-eq*)

show *?case*
proof
assume *finite* {*l* :: *'a list. length* (*tl l*) = *Suc y*}

then have *finite* (*tl* ` {*l* :: *'a list. length* (*tl l*) = *Suc y*})
by *blast*

moreover have *tl* ` {*l* :: *'a list. length* (*tl l*) = *Suc y*} = {*l* :: *'a list. length l* = *Suc y*}
using *2*
by *auto*

ultimately show *False*
using *Suc 1*
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong One-nat-def finite-insert*)
qed
qed

lemma *infinite-prods'*: {*p* :: *'a* × *'a* . *fst p* = *y*} = {*y*} × *UNIV*
by *auto*

lemma *infinite-prods*: *infinite* {*p* :: ((*'a* :: *infinite*) × *'a*). *fst p* = *y*}
unfolding *infinite-prods'*
using *finite-cartesian-productD2 infinite-UNIV* **by** *blast*

lemma *nat-version'*: ∃*f* :: *nat* ⇒ *nat*. ∀ *y* :: *nat*. *infinite* {*x*. *f x* = *y*}
proof−
obtain *g* :: *nat* ⇒ *nat* × *nat* **where** *bij-g*: *bij g*
using *bij-prod-decode* **by** *blast*

define *f* :: *nat* ⇒ *nat* **where**
⋀*x*. *f x* ≡ *fst* (*g x*)

have ⋀*y*. *infinite* {*x*. *f x* = *y*}
proof−
fix *y*
have *x*: {*x*. *fst* (*g x*) = *y*} = *inv g* ` {*p*. *fst p* = *y*}
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong bij-g bij-image-Collect-eq bij-imp-bij-inv inv-inv-eq*)

show *infinite* {*x*. *f x* = *y*}
unfolding *f-def x*
using *infinite-prods*
by (*metis bij-betw-def bij-g finite-imageI image-f-inv-f*)
qed

then show *?thesis*

by *blast*
qed

lemma *not-nat-version'*: $\exists f :: ('a :: infinite) \Rightarrow 'a.\ \forall y.\ infinite\ \{x.\ f\ x = y\}$
proof −
obtain $g :: 'a \Rightarrow 'a \times 'a$ **where** *bij-g*: *bij g*
using *Times-same-infinite-bij-betw-types bij-betw-inv infinite-UNIV* **by** *blast*

define $f :: 'a \Rightarrow 'a$ **where**
$\bigwedge x.\ f\ x \equiv fst\ (g\ x)$

have $\bigwedge y.\ infinite\ \{x.\ f\ x = y\}$
proof −
fix *y*
have *x*: $\{x.\ fst\ (g\ x) = y\} = inv\ g\ `\ \{p.\ fst\ p = y\}$
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong bij-g bij-image-Collect-eq bij-imp-bij-inv inv-inv-eq*)

show *infinite* $\{x.\ f\ x = y\}$
unfolding *f-def x*
using *infinite-prods*
by (*metis bij-g bij-is-surj finite-imageI image-f-inv-f*)
qed

then show *?thesis*
by *blast*
qed

lemma *not-nat-version''*:
assumes $|UNIV :: 'b\ set| \leq o\ |UNIV :: ('a :: infinite)\ set|$
shows $\exists f :: 'a \Rightarrow 'b.\ \forall y.\ infinite\ \{x.\ f\ x = y\}$
proof −
obtain $g :: 'a \Rightarrow 'a \times 'a$ **where** *bij-g*: *bij g*
using *Times-same-infinite-bij-betw-types bij-betw-inv infinite-UNIV* **by** *blast*

define $f :: 'a \Rightarrow 'a$ **where**
$\bigwedge x.\ f\ x \equiv fst\ (g\ x)$

have *inf*: $\bigwedge y.\ infinite\ \{x.\ f\ x = y\}$
proof −
fix *y*
have *x*: $\{x.\ fst\ (g\ x) = y\} = inv\ g\ `\ \{p.\ fst\ p = y\}$
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong bij-g bij-image-Collect-eq bij-imp-bij-inv inv-inv-eq*)

show *infinite* $\{x.\ f\ x = y\}$
unfolding *f-def x*
using *infinite-prods*
by (*metis bij-g bij-is-surj finite-imageI image-f-inv-f*)

qed

obtain $f' :: \ 'a \Rightarrow 'b$ **where** *surj* f'
using *assms*
by (*metis card-of-ordLeq2 empty-not-UNIV*)

then have $\bigwedge y.$ *infinite* $\{x.\ f'\ (f\ x) = y\}$
using *inf*
by (*smt* (*verit*, *ccfv-SIG*) *Collect-mono finite-subset surjD*)

then show *?thesis*
by *meson*
qed

lemma *nat-version*: $\exists f :: nat \Rightarrow nat.\ \forall y :: nat.$ *infinite* $\{x.\ f\ x = y\}$
proof−
obtain $g :: nat \Rightarrow nat\ list$ **where** *bij-g*: *bij g*
using *bij-list-decode* **by** *blast*

define $f :: nat \Rightarrow nat$ **where**
$\bigwedge x.\ f\ x \equiv length\ (tl\ (g\ x))$

have $\bigwedge y.$ *infinite* $\{x.\ f\ x = y\}$
proof−
fix y
have $\{x.\ length\ (tl\ (g\ x)) = y\} = inv\ g\ `\ \{l.\ length\ (tl\ l) = y\}$
by (*smt* (*verit*, *ccfv-SIG*) *Collect-cong bij-betw-def bij-g bij-image-Collect-eq*
image-inv-f-f
inv-inv-eq surj-imp-inj-inv)

then show *infinite* $\{x.\ f\ x = y\}$
unfolding *f-def*
using *infinite-lists-per-length*
by (*metis bij-g bij-is-surj finite-imageI image-f-inv-f*)
qed

then show *?thesis*
by *blast*
qed

definition *all-types* **where**
all-types $\mathcal{V} \equiv \forall\ ty.$ *infinite* $\{x.\ \mathcal{V}\ x = ty\}$

lemma *all-types-nat*: $\exists \mathcal{V} :: nat \Rightarrow nat.$ *all-types* $\mathcal{V}$
unfolding *all-types-def*
using *nat-version*

by *blast*

lemma *all-types*: $\exists\mathcal{V}$ :: (*'v* :: {*infinite*, *countable*} $\Rightarrow$ *'ty* :: *countable*). *all-types* $\mathcal{V}$
proof−
  obtain $\mathcal{V}$*-nat* :: *nat* $\Rightarrow$ *nat* **where** $\mathcal{V}$*-nat*: *all-types* $\mathcal{V}$*-nat*
    using *all-types-nat*
    by *blast*

  obtain *v-to-nat* :: *'v* $\Rightarrow$ *nat* **where** *v-to-nat*: *bij v-to-nat*
    using *countableI-type infinite-UNIV to-nat-on-infinite* **by** *blast*

  obtain *nat-to-ty* :: *nat* $\Rightarrow$ *'ty* **and** *N* **where** *nat-to-ty*: *bij-betw nat-to-ty N UNIV*
    using *countableE-bij*
    by (*metis countableI-type*)

  define $\mathcal{V}$ **where** $\bigwedge x$. $\mathcal{V}$ *x* $\equiv$ *nat-to-ty* ($\mathcal{V}$*-nat* (*v-to-nat x*))

  have *1*: $\bigwedge ty$. {*x*. $\mathcal{V}$*-nat* (*v-to-nat x*) = *ty*} = *inv v-to-nat* ` {*x*. $\mathcal{V}$*-nat x* = *ty*}
    by (*smt* (*verit*, *best*) *Collect-cong bij-image-Collect-eq bij-imp-bij-inv inv-inv-eq v-to-nat*)

  have *2*: $\bigwedge ty$. *infinite* {*x*. $\mathcal{V}$*-nat* (*v-to-nat x*) = *ty*}
    unfolding *1*
    using $\mathcal{V}$*-nat*
    unfolding *all-types-def*
    by (*metis bij-betw-def finite-imageI image-f-inv-f v-to-nat*)

  have $\bigwedge ty$. *infinite* {*x*. $\mathcal{V}$ *x* = *ty*}
    using $\mathcal{V}$*-nat*
    unfolding $\mathcal{V}$*-def all-types-def*
    by (*smt* (*verit*) *2 Collect-mono UNIV-I bij-betw-iff-bijections finite-subset nat-to-ty*)

  then show $\exists\mathcal{V}$ :: *'v* :: {*infinite*, *countable*} $\Rightarrow$ *'ty* :: *countable*. *all-types* $\mathcal{V}$
    unfolding *all-types-def*
    by *fast*
qed

lemma *all-types'*:
  assumes |*UNIV* :: *'ty set*| $\leq o$ |*UNIV* :: (*'v* :: *infinite*) *set*|
  shows $\exists\mathcal{V}$ :: (*'v* :: *infinite* $\Rightarrow$ *'ty*). *all-types* $\mathcal{V}$
  using *not-nat-version''*[*OF assms*]
  unfolding *all-types-def*
  by *argo*

definition *clause-groundings* :: (*'f*, *'ty*) *fun-types* $\Rightarrow$ (*'f*, *'v*, *'ty*) *typed-clause* $\Rightarrow$ *'f ground-atom clause set* **where**
  clause-groundings $\mathcal{F}$ *clause* = { *clause.to-ground* (*fst clause* $\cdot$ γ) | γ.
    term-subst.is-ground-subst γ $\wedge$

$welltyped_c$ $\mathcal{F}$ (*snd clause*) (*fst clause*) $\wedge$
$welltyped_\sigma$-*on* (*clause.vars* (*fst clause*)) $\mathcal{F}$ (*snd clause*) γ $\wedge$
all-types (*snd clause*)
}

abbreviation *select-subst-stability-on* **where**
$\bigwedge$*select* $select_G$. *select-subst-stability-on* $\mathcal{F}$ *select* $select_G$ *premises* $\equiv$
$\forall$ $premise_G$ $\in$ $\bigcup$ (*clause-groundings* $\mathcal{F}$ ‘ *premises*). $\exists$ (*premise*, $\mathcal{V}$) $\in$ *premises*. $\exists$ γ.
premise $\cdot$ γ = *clause.from-ground* $premise_G$ $\wedge$
$select_G$ (*clause.to-ground* (*premise* $\cdot$ γ)) = *clause.to-ground* ((*select premise*) $\cdot$ γ) $\wedge$
$welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ *premise* $\wedge$ $welltyped_\sigma$-*on* (*clause.vars premise*) $\mathcal{F}$ $\mathcal{V}$ γ $\wedge$
term-subst.is-ground-subst γ $\wedge$
all-types $\mathcal{V}$

lemma *obtain-subst-stable-on-select-grounding*:
fixes *select* :: ('*f*, '*v*) *select*
obtains $select_G$ **where**
select-subst-stability-on $\mathcal{F}$ *select* $select_G$ *premises*
is-select-grounding select $select_G$
proof−
let *?premise-groundings* = $\bigcup$(*clause-groundings* $\mathcal{F}$ ‘ *premises*)

have $select_G$-*exists-for-premises*:
$\forall$ $premise_G$ $\in$ *?premise-groundings*. $\exists$ $select_G$ γ. $\exists$ (*premise*, $\mathcal{V}$) $\in$ *premises*.
premise $\cdot$ γ = *clause.from-ground* $premise_G$
$\wedge$ $select_G$ $premise_G$ = *clause.to-ground* ((*select premise*) $\cdot$ γ)
$\wedge$ $welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ *premise* $\wedge$ $welltyped_\sigma$-*on* (*clause.vars premise*) $\mathcal{F}$ $\mathcal{V}$ γ
$\wedge$ *term-subst.is-ground-subst* γ $\wedge$ *all-types* $\mathcal{V}$
unfolding *clause-groundings-def*
using *clause.is-ground-subst-is-ground*
by *fastforce*

obtain $select_G$-*on-premise-groundings* **where**
$select_G$-*on-premise-groundings*: $\forall$ $premise_G$ $\in$*?premise-groundings*. $\exists$ (*premise*, $\mathcal{V}$) $\in$ *premises*. $\exists$ γ.
premise $\cdot$ γ = *clause.from-ground* $premise_G$
$\wedge$ $select_G$-*on-premise-groundings* (*clause.to-ground* (*premise* $\cdot$ γ)) =
clause.to-ground ((*select premise*) $\cdot$ γ)
$\wedge$ $welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ *premise* $\wedge$ $welltyped_\sigma$-*on* (*clause.vars premise*) $\mathcal{F}$ $\mathcal{V}$ γ
$\wedge$ *term-subst.is-ground-subst* γ $\wedge$ *all-types* $\mathcal{V}$
using *Ball-Ex-comm*(*1*)[*OF* $select_G$-*exists-for-premises*]
prod.case-eq-if clause.from-ground-inverse
by *fastforce*

define $select_G$ **where**
$\bigwedge$$clause_G$. $select_G$ $clause_G$ = (
if $clause_G$ $\in$ *?premise-groundings*

then $select_G$*-on-premise-groundings* $clause_G$
else clause.to-ground (*select* (*clause.from-ground* $clause_G$))
)

have *grounding*: *is-select-grounding select* $select_G$
proof−
have $\bigwedge$ $clause_G$ *a b*.
⟦∀ *y*∈*premises*.
∀ $premise_G$∈*clause-groundings* $\mathcal{F}$ *y*.
∃ *x*∈*premises*.
case x of
(*premise*, $\mathcal{V}$) ⇒
∃ γ. *premise* · γ = *clause.from-ground* $premise_G$ ∧
$select_G$*-on-premise-groundings* (*clause.to-ground* (*premise* · γ))
=
clause.to-ground (*select premise* · γ) ∧
$welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ *premise* ∧
$welltyped_\sigma$*-on* (*clause.vars premise*) $\mathcal{F}$ $\mathcal{V}$ γ ∧
term-subst.is-ground-subst γ ∧ *all-types* $\mathcal{V}$;
(*a*, *b*) ∈ *premises*; $clause_G$ ∈ *clause-groundings* $\mathcal{F}$ (*a*, *b*)⟧
⟹ ∃ *clause* γ.
clause.vars (*clause* · γ) = {} ∧
$clause_G$ = *clause.to-ground* (*clause* · γ) ∧
$select_G$*-on-premise-groundings* $clause_G$ = *clause.to-ground* (*select clause*
· γ)
by *force*

moreover have $\bigwedge$ $clause_G$.
⟦∀ *y*∈*premises*.
∀ $premise_G$∈*clause-groundings* $\mathcal{F}$ *y*.
∃ *x*∈*premises*.
case x of
(*premise*, $\mathcal{V}$) ⇒
∃ γ. *premise* · γ = *clause.from-ground* $premise_G$ ∧
$select_G$*-on-premise-groundings* (*clause.to-ground* (*premise* · γ))
=
clause.to-ground (*select premise* · γ) ∧
$welltyped_c$ $\mathcal{F}$ $\mathcal{V}$ *premise* ∧
$welltyped_\sigma$*-on* (*clause.vars premise*) $\mathcal{F}$ $\mathcal{V}$ γ ∧
term-subst.is-ground-subst γ ∧ *all-types* $\mathcal{V}$;
∀ *x*∈*premises*. $clause_G$ ∉ *clause-groundings* $\mathcal{F}$ *x*⟧
⟹ ∃ *clause* γ.
clause.vars (*clause* · γ) = {} ∧
$clause_G$ = *clause.to-ground* (*clause* · γ) ∧
clause.to-ground (*select* (*clause.from-ground* $clause_G$)) =
clause.to-ground (*select clause* · γ)
by (*metis* (*no-types*, *opaque-lifting*) *clause.comp-subst.left.action-neutral*
clause.ground-is-ground clause.from-ground-inverse)

ultimately show *?thesis*
unfolding *is-select-grounding-def* $select_G$*-def*
using $select_G$*-on-premise-groundings*
by *auto*
qed

show *?thesis*
using *that*[*OF* - *grounding*] $select_G$*-on-premise-groundings*
unfolding $select_G$*-def*
by *fastforce*
qed

locale *first-order-select* = *select select*
for *select* :: (′*f*, ′*v*) *atom clause* ⇒ (′*f*, ′*v*) *atom clause*
begin

abbreviation *is-grounding* :: ′*f ground-select* ⇒ *bool* **where**
is-grounding $select_G$ ≡ *is-select-grounding select* $select_G$

definition $select_{Gs}$ **where**
$select_{Gs}$ = { *ground-select*. *is-grounding ground-select* }

definition $select_G$*-simple* **where**
$select_G$*-simple clause* = *clause.to-ground* (*select* (*clause.from-ground clause*))

lemma $select_G$*-simple*: *is-grounding* $select_G$*-simple*
unfolding *is-select-grounding-def* $select_G$*-simple-def*
by (*metis clause.from-ground-inverse clause.ground-is-ground clause.subst-id-subst*)

lemma *select-from-ground-clause1*:
assumes *clause.is-ground clause*
shows *clause.is-ground* (*select clause*)
using *select-subset sub-ground-clause assms*
by *metis*

lemma *select-from-ground-clause2*:
assumes *literal* ∈# *select* (*clause.from-ground clause*)
shows *literal.is-ground literal*
using *assms clause.sub-in-ground-is-ground select-subset*
by *blast*

lemma *select-from-ground-clause3*:
assumes *clause.is-ground clause* $literal_G$ ∈# *clause.to-ground clause*
shows *literal.from-ground* $literal_G$ ∈# *clause*
using *assms*
by (*metis clause.to-ground-inverse clause.ground-sub-in-ground*)

lemmas *select-from-ground-clause* =
select-from-ground-clause1

select-from-ground-clause2
select-from-ground-clause3

lemma *select-subst1*:
assumes *clause.is-ground* (*clause* · γ)
shows *clause.is-ground* (*select clause* · γ)
using *assms*
by (*metis image-mset-subseteq-mono select-subset sub-ground-clause clause.subst-def*)

lemma *select-subst2*:
assumes *literal* ∈# *select clause* · γ
shows *is-neg literal*
using *assms subst-neg-stable select-negative-lits*
unfolding *clause.subst-def*
by *auto*

lemmas *select-subst* = *select-subst1 select-subst2*

end

locale *grounded-first-order-select* =
first-order-select select **for** *select* +
fixes $select_G$
assumes $select_G$: *is-select-grounding select* $select_G$
begin

abbreviation *subst-stability-on* **where**
subst-stability-on $\mathcal{F}$ *premises* ≡ *select-subst-stability-on* $\mathcal{F}$ *select* $select_G$ *premises*

lemma $select_G$*-subset*: $select_G$ *clause* ⊆# *clause*
using $select_G$
unfolding *is-select-grounding-def*
by (*metis select-subset clause.to-ground-def image-mset-subseteq-mono clause.subst-def*)

lemma $select_G$*-negative*:
assumes $literal_G$ ∈# $select_G$ $clause_G$
shows *is-neg* $literal_G$
proof −
obtain *clause* γ **where**
is-ground: *clause.is-ground* (*clause* · γ) **and**
$select_G$: $select_G$ $clause_G$ = *clause.to-ground* (*select clause* · γ)
using $select_G$
unfolding *is-select-grounding-def*
by *blast*

show *?thesis*
using
select-from-ground-clause(*3*)[
OF select-subst(*1*)[*OF is-ground*] *assms*[*unfolded* $select_G$],

THEN select-subst(2)
]
unfolding *literal.from-ground-def*
by *simp*
qed

sublocale *ground*: *select* $select_G$
by *unfold-locales* (*simp-all add*: $select_G$*-subset* $select_G$*-negative*)

end

end
theory *First-Order-Ordering*
imports
First-Order-Clause
Ground-Ordering
Relation-Extra
begin

context *ground-ordering*
begin

lemmas $less_{lG}$*-transitive-on* = *literal-order.transp-on-less*
lemmas $less_{lG}$*-asymmetric-on* = *literal-order.asymp-on-less*
lemmas $less_{lG}$*-total-on* = *literal-order.totalp-on-less*

lemmas $less_{cG}$*-transitive-on* = *clause-order.transp-on-less*
lemmas $less_{cG}$*-asymmetric-on* = *clause-order.asymp-on-less*
lemmas $less_{cG}$*-total-on* = *clause-order.totalp-on-less*

lemmas *is-maximal-lit-def* = *is-maximal-in-mset-wrt-iff*[*OF* $less_{lG}$*-transitive-on* $less_{lG}$*-asymmetric-on*]
lemmas *is-strictly-maximal-lit-def* =
is-strictly-maximal-in-mset-wrt-iff[*OF* $less_{lG}$*-transitive-on* $less_{lG}$*-asymmetric-on*]

end

6 First order ordering

locale *first-order-ordering* = *term-ordering-lifting* $less_t$
for
$less_t$ :: ('f, 'v) *term* ⇒ ('f, 'v) *term* ⇒ *bool* (**infix** $\prec_t$ *50*) +
assumes
$less_t$*-total-on* [*intro*]: *totalp-on* {*term. term.is-ground term*} ($\prec_t$) **and**
$less_t$*-wellfounded-on*: *wfp-on* {*term. term.is-ground term*} ($\prec_t$) **and**
$less_t$*-ground-context-compatible*:
⋀*context* $term_1$ $term_2$.
$term_1 \prec_t term_2 \Longrightarrow$
term.is-ground $term_1 \Longrightarrow$

term.is-ground $term_2 \Longrightarrow$
context.is-ground context $\Longrightarrow$
$context\langle term_1\rangle \prec_t context\langle term_2\rangle$ **and**
$less_t$*-ground-subst-stability*:
$\bigwedge term_1\ term_2\ (\gamma :: 'v \Rightarrow ('f,\ 'v)\ term).$
term.is-ground $(term_1 \cdot t\ \gamma) \Longrightarrow$
term.is-ground $(term_2 \cdot t\ \gamma) \Longrightarrow$
$term_1 \prec_t term_2 \Longrightarrow$
$term_1 \cdot t\ \gamma \prec_t term_2 \cdot t\ \gamma$ **and**
$less_t$*-ground-subterm-property*:
$\bigwedge term_G\ context_G.$
term.is-ground $term_G \Longrightarrow$
context.is-ground $context_G \Longrightarrow$
$context_G \neq \square \Longrightarrow$
$term_G \prec_t context_G\langle term_G\rangle$
begin

lemmas $less_t$*-transitive* = *transp-less-trm*
lemmas $less_t$*-asymmetric* = *asymp-less-trm*

6.1 Definitions

abbreviation $less\text{-}eq_t$ (**infix** $\preceq_t$ *50*) **where**
$less\text{-}eq_t \equiv (\prec_t)^{==}$

definition $less_{tG}$ :: *'f ground-term* ⇒ *'f ground-term* ⇒ *bool* (**infix** $\prec_{tG}$ *50*) **where**
$term_{G1} \prec_{tG} term_{G2} \equiv$ *term.from-ground* $term_{G1} \prec_t$ *term.from-ground* $term_{G2}$

notation *less-lit* (**infix** $\prec_l$ *50*)
notation *less-cls* (**infix** $\prec_c$ *50*)

lemma
assumes
L-in: $L \in\# C$ **and**
subst-stability: $\bigwedge L\ K.\ L \prec_l K \Longrightarrow (L \cdot l\ \sigma) \prec_l (K \cdot l\ \sigma)$ **and**
Lσ-max-in-Cσ: *literal-order.is-maximal-in-mset* $(C \cdot \sigma)\ (L \cdot l\ \sigma)$
shows *literal-order.is-maximal-in-mset C L*
proof −
have *Lσ-in*: $L \cdot l\ \sigma \in\# C \cdot \sigma$ **and** *Lσ-max*: $\forall y\in\# C \cdot \sigma.\ y \neq L \cdot l\ \sigma \longrightarrow \neg L \cdot l$
$\sigma \prec_l y$
using *Lσ-max-in-Cσ*
unfolding *atomize-conj literal-order.is-maximal-in-mset-iff*
by *argo*

show *literal-order.is-maximal-in-mset C L*
unfolding *literal-order.is-maximal-in-mset-iff*
proof (*intro conjI ballI impI*)
show $L \in\# C$
using *L-in* **.**

next
  show $\bigwedge y.\ y \in\# C \Longrightarrow y \neq L \Longrightarrow \neg L \prec_l y$
    using *subst-stability*
  by (*metis Lσ-max clause.subst-in-to-set-subst literal-order.order.strict-iff-order*)
qed
qed

lemmas $less_l$*-def* = *less-lit-def*
lemmas $less_c$*-def* = *less-cls-def*

abbreviation *less-eq*$_l$ (**infix** $\preceq_l$ *50*) **where**
less-eq$_l$ ≡ $(\prec_l)^{==}$

abbreviation *less-eq*$_c$ (**infix** $\preceq_c$ *50*) **where**
less-eq$_c$ ≡ $(\prec_c)^{==}$

abbreviation *is-maximal*$_l$ ::
('f, 'v) *atom literal* ⇒ ('f, 'v) *atom clause* ⇒ *bool* **where**
is-maximal$_l$ *literal clause* ≡ *is-maximal-in-mset-wrt* $(\prec_l)$ *clause literal*

abbreviation *is-strictly-maximal*$_l$ ::
('f, 'v) *atom literal* ⇒ ('f, 'v) *atom clause* ⇒ *bool* **where**
is-strictly-maximal$_l$ *literal clause* ≡ *is-strictly-maximal-in-mset-wrt* $(\prec_l)$ *clause literal*

6.2 Term ordering

lemmas $less_t$*-asymmetric-on* = *term-order.asymp-on-less*
lemmas $less_t$*-irreflexive-on* = *term-order.irreflp-on-less*
lemmas $less_t$*-transitive-on* = *term-order.transp-on-less*

lemma $less_t$*-wellfounded-on'*: *Wellfounded.wfp-on* (*term.from-ground* ‘ $terms_G$) $(\prec_t)$
proof (*rule Wellfounded.wfp-on-subset*)
  show *Wellfounded.wfp-on* {*term. term.is-ground term*} $(\prec_t)$
    using $less_t$*-wellfounded-on* **.**
next
  show *term.from-ground* ‘ $terms_G$ ⊆ {*term. term.is-ground term*}
    by *force*
qed

lemma $less_t$*-total-on'*: *totalp-on* (*term.from-ground* ‘ $terms_G$) $(\prec_t)$
  using $less_t$*-total-on*
  by (*simp add*: *totalp-on-def*)

lemma $less_{tG}$*-wellfounded*: *wfp* $(\prec_{tG})$
proof −
  have *Wellfounded.wfp-on* (*range term.from-ground*) $(\prec_t)$
    using $less_t$*-wellfounded-on'* **by** *metis*

hence *wfp* $(\lambda term_{G1}\ term_{G2}.\ term.from\text{-}ground\ term_{G1} \prec_t term.from\text{-}ground\ term_{G2})$
unfolding *Wellfounded.wfp-on-image*[*symmetric*] **.**
thus *wfp* $(\prec_{tG})$
unfolding $less_{tG}$*-def* **.**
qed

6.3 Ground term ordering

lemma $less_{tG}$*-asymmetric* [*intro*]: *asymp* $(\prec_{tG})$
by (*simp add*: *wfP-imp-asymp* $less_{tG}$*-wellfounded*)

lemmas $less_{tG}$*-asymmetric-on* = $less_{tG}$*-asymmetric*[*THEN asymp-on-subset*, *OF subset-UNIV*]

lemma $less_{tG}$*-transitive* [*intro*]: *transp* $(\prec_{tG})$
using $less_{tG}$*-def* $less_t$*-transitive transpE transpI*
by (*metis* (*full-types*))

lemmas $less_{tG}$*-transitive-on* = $less_{tG}$*-transitive*[*THEN transp-on-subset*, *OF subset-UNIV*]

lemma $less_{tG}$*-total* [*intro*]: *totalp* $(\prec_{tG})$
unfolding $less_{tG}$*-def*
using *totalp-on-image*[*OF inj-term-of-gterm*] $less_t$*-total-on'*
by *blast*

lemmas $less_{tG}$*-total-on* = $less_{tG}$*-total*[*THEN totalp-on-subset*, *OF subset-UNIV*]

lemma $less_{tG}$*-context-compatible* [*simp*]:
assumes $term_1 \prec_{tG} term_2$
shows $context\langle term_1\rangle_G \prec_{tG} context\langle term_2\rangle_G$
using *assms* $less_t$*-ground-context-compatible*
unfolding $less_{tG}$*-def*
by (*metis context.ground-is-ground term.ground-is-ground ground-term-with-context*(*3*))

lemma $less_{tG}$*-subterm-property* [*simp*]:
assumes $context \neq \Box_G$
shows $term \prec_{tG} context\langle term\rangle_G$
using
assms
$less_t$*-ground-subterm-property*[*OF term.ground-is-ground context.ground-is-ground*]

context-from-ground-hole
unfolding $less_{tG}$*-def ground-term-with-context*(*3*)
by *blast*

lemma $less_t$*-*$less_{tG}$ [*clause-simp*]:

assumes *term.is-ground* $term_1$ **and** *term.is-ground* $term_2$
shows $term_1 \prec_t term_2 \longleftrightarrow$ *term.to-ground* $term_1 \prec_{tG}$ *term.to-ground* $term_2$
by (*simp add*: *assms* $less_{tG}$*-def*)

lemma *less-eq*$_t$*-ground-subst-stability*:
assumes *term.is-ground* $(term_1 \cdot t\ \gamma)$ *term.is-ground* $(term_2 \cdot t\ \gamma)$ $term_1 \preceq_t term_2$
shows $term_1 \cdot t\ \gamma \preceq_t term_2 \cdot t\ \gamma$
using *less*$_t$*-ground-subst-stability*[*OF assms*(*1*, *2*)] *assms*(*3*)
by *auto*

6.4 Literal ordering

lemmas *less*$_l$*-asymmetric* [*intro*] = *literal-order.asymp-on-less*[*of UNIV*]
lemmas *less*$_l$*-asymmetric-on* [*intro*] = *literal-order.asymp-on-less*

lemmas *less*$_l$*-transitive* [*intro*] = *literal-order.transp-on-less*[*of UNIV*]
lemmas *less*$_l$*-transitive-on* = *literal-order.transp-on-less*

lemmas *is-maximal*$_l$*-def* = *is-maximal-in-mset-wrt-iff*[*OF less*$_l$*-transitive-on less*$_l$*-asymmetric-on*]

lemmas *is-strictly-maximal*$_l$*-def* =
is-strictly-maximal-in-mset-wrt-iff[*OF less*$_l$*-transitive-on less*$_l$*-asymmetric-on*]

lemmas *is-maximal*$_l$*-if-is-strictly-maximal*$_l$ =
is-maximal-in-mset-wrt-if-is-strictly-maximal-in-mset-wrt[*OF*
less$_l$*-transitive-on less*$_l$*-asymmetric-on*
]

lemma *less*$_l$*-ground-subst-stability*:
assumes
literal.is-ground (*literal* $\cdot l\ \gamma$)
literal.is-ground (*literal'* $\cdot l\ \gamma$)
shows *literal* $\prec_l$ *literal'* $\Longrightarrow$ *literal* $\cdot l\ \gamma \prec_l$ *literal'* $\cdot l\ \gamma$
unfolding *less*$_l$*-def mset-mset-lit-subst*[*symmetric*]
proof (*elim multp-map-strong*[*rotated* -1])
show *monotone-on* (*set-mset* (*mset-lit literal* + *mset-lit literal'*)) $(\prec_t)$ $(\prec_t)$ $(\lambda term.\ term \cdot t\ \gamma)$
by (*rule monotone-onI*)
(*metis assms*(*1*,*2*) *less*$_t$*-ground-subst-stability ground-term-in-ground-literal-subst union-iff*)
qed (*use less*$_t$*-asymmetric less*$_t$*-transitive* **in** *simp-all*)

lemma *maximal*$_l$*-in-clause*:
assumes *is-maximal*$_l$ *literal clause*
shows *literal* $\in\#$ *clause*
using *assms*
unfolding *is-maximal*$_l$*-def*
by(*rule conjunct1*)

lemma *strictly-maximal*$_l$*-in-clause*:
  assumes *is-strictly-maximal*$_l$ *literal clause*
  shows *literal* $\in\#$ *clause*
  using *assms*
  unfolding *is-strictly-maximal*$_l$*-def*
  by(*rule conjunct1*)

6.5 Clause ordering

lemmas *less*$_c$*-asymmetric* [*intro*] = *clause-order.asymp-on-less*[*of UNIV*]
lemmas *less*$_c$*-asymmetric-on* [*intro*] = *clause-order.asymp-on-less*
lemmas *less*$_c$*-transitive* [*intro*] = *clause-order.transp-on-less*[*of UNIV*]
lemmas *less*$_c$*-transitive-on* [*intro*] = *clause-order.transp-on-less*

lemma *less*$_c$*-ground-subst-stability*:
  assumes
    clause.is-ground (*clause* $\cdot$ γ)
    clause.is-ground (*clause'* $\cdot$ γ)
  shows *clause* $\prec_c$ *clause'* $\Longrightarrow$ *clause* $\cdot$ γ $\prec_c$ *clause'* $\cdot$ γ
  unfolding *clause.subst-def less*$_c$*-def*
proof (*elim multp-map-strong*[*rotated* -1])
  show *monotone-on* (*set-mset* (*clause* + *clause'*)) ($\prec_l$) ($\prec_l$) (λ*literal. literal* $\cdot l$ γ)
    by (*rule monotone-onI*)
        (*metis assms*(*1*,*2*) *clause.to-set-is-ground-subst less*$_l$*-ground-subst-stability*
union-iff)
qed (*use less*$_l$*-asymmetric less*$_l$*-transitive* **in** *simp-all*)

6.6 Grounding

sublocale *ground*: *ground-ordering* ($\prec_{tG}$)
  apply *unfold-locales*
  by(*simp-all add*: *less*$_{tG}$*-transitive less*$_{tG}$*-asymmetric less*$_{tG}$*-wellfounded less*$_{tG}$*-total*)

notation *ground.less-lit* (**infix** $\prec_{lG}$ *50*)
notation *ground.less-cls* (**infix** $\prec_{cG}$ *50*)

notation *ground.lesseq-trm* (**infix** $\preceq_{tG}$ *50*)
notation *ground.lesseq-lit* (**infix** $\preceq_{lG}$ *50*)
notation *ground.lesseq-cls* (**infix** $\preceq_{cG}$ *50*)

lemma *not-less-eq*$_{tG}$: $\neg$ *term*$_{G2}$ $\preceq_{tG}$ *term*$_{G1}$ $\longleftrightarrow$ *term*$_{G1}$ $\prec_{tG}$ *term*$_{G2}$
  using *ground.term-order.not-le* **.**

lemma *less-eq*$_t$*-less-eq*$_{tG}$:
  assumes *term.is-ground term*$_1$ **and** *term.is-ground term*$_2$
  shows *term*$_1$ $\preceq_t$ *term*$_2$ $\longleftrightarrow$ *term.to-ground term*$_1$ $\preceq_{tG}$ *term.to-ground term*$_2$
  unfolding *reflclp-iff less*$_t$*-less*$_{tG}$[*OF assms*]
  using *assms*[*THEN term.to-ground-inverse*]
  by *auto*

lemma *less-eq*$_{tG}$*-less-eq*$_t$:
term$_{G1}$ $\preceq_{tG}$ *term*$_{G2}$ $\longleftrightarrow$ *term.from-ground term*$_{G1}$ $\preceq_t$ *term.from-ground term*$_{G2}$
unfolding
less-eq$_t$*-less-eq*$_{tG}$[*OF term.ground-is-ground term.ground-is-ground*]
term.from-ground-inverse
..

lemma *not-less-eq*$_t$:
assumes *term.is-ground term*$_1$ **and** *term.is-ground term*$_2$
shows $\neg$ *term*$_2$ $\preceq_t$ *term*$_1$ $\longleftrightarrow$ *term*$_1$ $\prec_t$ *term*$_2$
unfolding *less*$_t$*-less*$_{tG}$[*OF assms*] *less-eq*$_t$*-less-eq*$_{tG}$[*OF assms*(*2*, *1*)] *not-less-eq*$_{tG}$
..

lemma *less*$_{lG}$*-less*$_l$:
literal$_{G1}$ $\prec_{lG}$ *literal*$_{G2}$ $\longleftrightarrow$ *literal.from-ground literal*$_{G1}$ $\prec_l$ *literal.from-ground literal*$_{G2}$
unfolding *less*$_l$*-def ground.less-lit-def less*$_{tG}$*-def mset-literal-from-ground*
using
multp-image-mset-image-msetI[*OF - less*$_t$*-transitive*]
multp-image-mset-image-msetD[*OF - less*$_t$*-transitive-on term.inj-from-ground*]
by *blast*

lemma *less*$_l$*-less*$_{lG}$:
assumes *literal.is-ground literal*$_1$ *literal.is-ground literal*$_2$
shows *literal*$_1$ $\prec_l$ *literal*$_2$ $\longleftrightarrow$ *literal.to-ground literal*$_1$ $\prec_{lG}$ *literal.to-ground literal*$_2$
using *assms*
by (*simp add*: *less*$_{lG}$*-less*$_l$)

lemma *less-eq*$_l$*-less-eq*$_{lG}$:
assumes *literal.is-ground literal*$_1$ **and** *literal.is-ground literal*$_2$
shows *literal*$_1$ $\preceq_l$ *literal*$_2$ $\longleftrightarrow$ *literal.to-ground literal*$_1$ $\preceq_{lG}$ *literal.to-ground literal*$_2$
unfolding *reflclp-iff less*$_l$*-less*$_{lG}$[*OF assms*]
using *assms*[*THEN literal.to-ground-inverse*]
by *auto*

lemma *less-eq*$_{lG}$*-less-eq*$_l$:
literal$_{G1}$ $\preceq_{lG}$ *literal*$_{G2}$ $\longleftrightarrow$ *literal.from-ground literal*$_{G1}$ $\preceq_l$ *literal.from-ground literal*$_{G2}$
unfolding
less-eq$_l$*-less-eq*$_{lG}$[*OF literal.ground-is-ground literal.ground-is-ground*]
literal.from-ground-inverse
..

lemma *maximal-lit-in-clause*:
assumes *ground.is-maximal-lit literal*$_G$ *clause*$_G$
shows *literal*$_G$ $\in\#$ *clause*$_G$

using *assms*
unfolding *ground.is-maximal-lit-def*
by(*rule conjunct1*)

lemma *is-maximal*$_l$*-empty* [*simp*]:
assumes *is-maximal*$_l$ *literal* {#}
shows *False*
using *assms maximal*$_l$*-in-clause*
by *fastforce*

lemma *is-strictly-maximal*$_l$*-empty* [*simp*]:
assumes *is-strictly-maximal*$_l$ *literal* {#}
shows *False*
using *assms strictly-maximal*$_l$*-in-clause*
by *fastforce*

lemma *is-maximal-lit-iff-is-maximal*$_l$:
ground.is-maximal-lit literal$_G$ *clause*$_G$ $\longleftrightarrow$
is-maximal$_l$ (*literal.from-ground literal*$_G$) (*clause.from-ground clause*$_G$)
unfolding
is-maximal$_l$*-def*
ground.is-maximal-lit-def
clause.ground-sub-in-ground[*symmetric*]
using
less$_l$*-less*$_{lG}$[*OF literal.ground-is-ground clause.sub-in-ground-is-ground*]
clause.sub-in-ground-is-ground
clause.ground-sub-in-ground
by (*metis literal.to-ground-inverse literal.from-ground-inverse*)

lemma *is-strictly-maximal*$_{Gl}$*-iff-is-strictly-maximal*$_l$:
ground.is-strictly-maximal-lit literal$_G$ *clause*$_G$
$\longleftrightarrow$ *is-strictly-maximal*$_l$ (*literal.from-ground literal*$_G$) (*clause.from-ground clause*$_G$)
unfolding
is-strictly-maximal-in-mset-wrt-iff-is-greatest-in-mset-wrt[*OF*
ground.less$_{lG}$*-transitive-on ground.less*$_{lG}$*-asymmetric-on ground.less*$_{lG}$*-total-on*,
symmetric
]
ground.is-strictly-maximal-lit-def
is-strictly-maximal$_l$*-def*
clause.ground-sub-in-ground[*symmetric*]
remove1-mset-literal-from-ground
clause.ground-sub
less-eq$_{lG}$*-less-eq*$_l$
..

lemma *not-less-eq*$_{lG}$: $\neg$ *literal*$_{G2}$ $\preceq_{lG}$ *literal*$_{G1}$ $\longleftrightarrow$ *literal*$_{G1}$ $\prec_{lG}$ *literal*$_{G2}$
using *asympD*[*OF ground.less*$_{lG}$*-asymmetric-on*] *totalpD*[*OF ground.less*$_{lG}$*-total-on*]
by *blast*

lemma *not-less-eq*$_l$:
assumes *literal.is-ground literal*$_1$ **and** *literal.is-ground literal*$_2$
shows $\neg$ *literal*$_2$ $\preceq_l$ *literal*$_1$ $\longleftrightarrow$ *literal*$_1$ $\prec_l$ *literal*$_2$
unfolding *less*$_l$*-less*$_{lG}$[*OF assms*] *less-eq*$_l$*-less-eq*$_{lG}$[*OF assms*(*2*, *1*)] *not-less-eq*$_{lG}$
..

lemma *less*$_{cG}$*-less*$_c$:
clause$_{G1}$ $\prec_{cG}$ *clause*$_{G2}$ $\longleftrightarrow$ *clause.from-ground clause*$_{G1}$ $\prec_c$ *clause.from-ground clause*$_{G2}$
proof (*rule iffI*)
show *clause*$_{G1}$ $\prec_{cG}$ *clause*$_{G2}$ $\Longrightarrow$ *clause.from-ground clause*$_{G1}$ $\prec_c$ *clause.from-ground clause*$_{G2}$
unfolding *less*$_c$*-def*
by (*auto simp*: *clause.from-ground-def ground.less-cls-def less*$_{lG}$*-less*$_l$
intro!: *multp-image-mset-image-msetI elim*: *multp-mono-strong*)
next
have *transp* ($\lambda x\ y$. *literal.from-ground x* $\prec_l$ *literal.from-ground y*)
by (*metis* (*no-types*, *lifting*) *literal-order.less-trans transpI*)
thus *clause.from-ground clause*$_{G1}$ $\prec_c$ *clause.from-ground clause*$_{G2}$ $\Longrightarrow$ *clause*$_{G1}$ $\prec_{cG}$ *clause*$_{G2}$
unfolding *ground.less-cls-def clause.from-ground-def less*$_c$*-def*
by (*auto simp*: *less*$_{lG}$*-less*$_l$
dest!: *multp-image-mset-image-msetD*[*OF - less*$_l$*-transitive literal.inj-from-ground*]
elim!: *multp-mono-strong*)
qed

lemma *less*$_c$*-less*$_{cG}$:
assumes *clause.is-ground clause*$_1$ *clause.is-ground clause*$_2$
shows *clause*$_1$ $\prec_c$ *clause*$_2$ $\longleftrightarrow$ *clause.to-ground clause*$_1$ $\prec_{cG}$ *clause.to-ground clause*$_2$
using *assms*
by (*simp add*: *less*$_{cG}$*-less*$_c$)

lemma *less-eq*$_c$*-less-eq*$_{cG}$:
assumes *clause.is-ground clause*$_1$ **and** *clause.is-ground clause*$_2$
shows *clause*$_1$ $\preceq_c$ *clause*$_2$ $\longleftrightarrow$ *clause.to-ground clause*$_1$ $\preceq_{cG}$ *clause.to-ground clause*$_2$
unfolding *reflclp-iff less*$_c$*-less*$_{cG}$[*OF assms*]
using *assms*[*THEN clause.to-ground-inverse*]
by *fastforce*

lemma *less-eq*$_{cG}$*-less-eq*$_c$:
clause$_{G1}$ $\preceq_{cG}$ *clause*$_{G2}$ $\longleftrightarrow$ *clause.from-ground clause*$_{G1}$ $\preceq_c$ *clause.from-ground clause*$_{G2}$
unfolding
less-eq$_c$*-less-eq*$_{cG}$[*OF clause.ground-is-ground clause.ground-is-ground*]
clause.from-ground-inverse
..

lemma *not-less-eq*$_{cG}$: ¬ *clause*$_{G2}$ $\preceq_{cG}$ *clause*$_{G1}$ ⟷ *clause*$_{G1}$ $\prec_{cG}$ *clause*$_{G2}$
using *asympD*[*OF ground.less*$_{cG}$*-asymmetric-on*] *totalpD*[*OF ground.less*$_{cG}$*-total-on*]
by *blast*

lemma *not-less-eq*$_c$:
assumes *clause.is-ground clause*$_1$ **and** *clause.is-ground clause*$_2$
shows ¬ *clause*$_2$ $\preceq_c$ *clause*$_1$ ⟷ *clause*$_1$ $\prec_c$ *clause*$_2$
unfolding *less*$_c$*-less*$_{cG}$[*OF assms*] *less-eq*$_c$*-less-eq*$_{cG}$[*OF assms*(*2*, *1*)] *not-less-eq*$_{cG}$
..

lemma *less*$_t$*-ground-context-compatible*′:
assumes
context.is-ground context
term.is-ground term
term.is-ground term′
context⟨*term*⟩ $\prec_t$ *context*⟨*term*′⟩
shows *term* $\prec_t$ *term*′
using *assms*
by (*metis less*$_t$*-ground-context-compatible not-less-eq*$_t$ *term-order.dual-order.asym*

term-order.order.not-eq-order-implies-strict)

lemma *less*$_t$*-ground-context-compatible-iff*:
assumes
context.is-ground context
term.is-ground term
term.is-ground term′
shows *context*⟨*term*⟩ $\prec_t$ *context*⟨*term*′⟩ ⟷ *term* $\prec_t$ *term*′
using *assms less*$_t$*-ground-context-compatible less*$_t$*-ground-context-compatible*′
by *blast*

6.7 Stability under ground substitution

lemma *less*$_t$*-less-eq*$_t$*-ground-subst-stability*:
assumes
term.is-ground (*term*$_1$ ·*t* γ)
term.is-ground (*term*$_2$ ·*t* γ)
term$_1$ ·*t* γ $\prec_t$ *term*$_2$ ·*t* γ
shows
¬ *term*$_2$ $\preceq_t$ *term*$_1$
proof
assume *assumption*: *term*$_2$ $\preceq_t$ *term*$_1$

have *term*$_2$ ·*t* γ $\preceq_t$ *term*$_1$ ·*t* γ
using *less-eq*$_t$*-ground-subst-stability*[*OF*
assms(*2*, *1*)
assumption
].

then show *False*
using *assms*(*3*) **by** *order*
qed

lemma *less-eq$_l$-ground-subst-stability*:
assumes
literal.is-ground (*literal*$_1$ ·*l* γ)
literal.is-ground (*literal*$_2$ ·*l* γ)
literal$_1$ $\preceq_l$ *literal*$_2$
shows *literal*$_1$ ·*l* γ $\preceq_l$ *literal*$_2$ ·*l* γ
using *less$_l$-ground-subst-stability*[*OF assms*(*1*, *2*)] *assms*(*3*)
by *auto*

lemma *less$_l$-less-eq$_l$-ground-subst-stability*: **assumes**
literal.is-ground (*literal*$_1$ ·*l* γ)
literal.is-ground (*literal*$_2$ ·*l* γ)
literal$_1$ ·*l* γ $\prec_l$ *literal*$_2$ ·*l* γ
shows
¬ *literal*$_2$ $\preceq_l$ *literal*$_1$
by (*meson assms less-eq$_l$-ground-subst-stability not-less-eq$_l$*)

lemma *less-eq$_c$-ground-subst-stability*:
assumes
clause.is-ground (*clause*$_1$ · γ)
clause.is-ground (*clause*$_2$ · γ)
clause$_1$ $\preceq_c$ *clause*$_2$
shows *clause*$_1$ · γ $\preceq_c$ *clause*$_2$ · γ
using *less$_c$-ground-subst-stability*[*OF assms*(*1*, *2*)] *assms*(*3*)
by *auto*

lemma *less$_c$-less-eq$_c$-ground-subst-stability*: **assumes**
clause.is-ground (*clause*$_1$ · γ)
clause.is-ground (*clause*$_2$ · γ)
clause$_1$ · γ $\prec_c$ *clause*$_2$ · γ
shows
¬ *clause*$_2$ $\preceq_c$ *clause*$_1$
by (*meson assms less-eq$_c$-ground-subst-stability not-less-eq$_c$*)

lemma *is-maximal$_l$-ground-subst-stability*:
assumes
clause-not-empty: *clause* $\neq$ {#} **and**
clause-grounding: *clause.is-ground* (*clause* · γ)
obtains *literal*
where *is-maximal$_l$ literal clause is-maximal$_l$* (*literal* ·*l* γ) (*clause* · γ)
proof−
assume *assumption*:
$\bigwedge$*literal*. *is-maximal$_l$ literal clause* $\Longrightarrow$ *is-maximal$_l$* (*literal* ·*l* γ) (*clause* · γ)
$\Longrightarrow$ *thesis*

from *clause-not-empty*
have *clause-grounding-not-empty*: *clause* · $\gamma \neq \{\#\}$
unfolding *clause.subst-def*
by *simp*

obtain *literal* **where**
literal: *literal* $\in\#$ *clause* **and**
literal-grounding-is-maximal: *is-maximal*$_l$ (*literal* ·*l* γ) (*clause* · γ)
using
ex-maximal-in-mset-wrt[*OF less*$_l$*-transitive-on less*$_l$*-asymmetric-on clause-grounding-not-empty*]

maximal$_l$*-in-clause*
unfolding *clause.subst-def*
by *force*

from *literal-grounding-is-maximal*
have *no-bigger-than-literal*:
$\forall$ *literal'* $\in\#$ *clause* · γ. *literal'* $\neq$ *literal* ·*l* $\gamma \longrightarrow \neg$ *literal* ·*l* $\gamma \prec_l$ *literal'*
unfolding *is-maximal*$_l$*-def*
by *simp*

show *?thesis*
proof(*cases is-maximal*$_l$ *literal clause*)
case *True*
with *literal-grounding-is-maximal assumption* **show** *?thesis*
by *blast*
next
case *False*
then obtain *literal'* **where**
literal': *literal'* $\in\#$ *clause literal* $\prec_l$ *literal'*
unfolding *is-maximal*$_l$*-def*
using *literal*
by *blast*

note *literals-in-clause* = *literal*(*1*) *literal'*(*1*)
note *literals-grounding* = *literals-in-clause*[*THEN*
clause.to-set-is-ground-subst[*OF - clause-grounding*]
]

have *literal* ·*l* $\gamma \prec_l$ *literal'* ·*l* γ
using *less*$_l$*-ground-subst-stability*[*OF literals-grounding literal'*(*2*)]**.**

then have *False*
using
no-bigger-than-literal
clause.subst-in-to-set-subst[*OF literal'*(*1*)]
by (*metis asymp-onD less*$_l$*-asymmetric-on*)

```
    then show ?thesis..
  qed
qed

lemma is-maximal_l-ground-subst-stability':
  assumes
   literal ∈# clause
   clause.is-ground (clause · γ)
   is-maximal_l (literal ·l γ) (clause · γ)
 shows
   is-maximal_l literal clause
proof(rule ccontr)
  assume ¬ is-maximal_l literal clause

  then obtain literal' where literal':
      literal ≺_l literal'
      literal' ∈# clause
  using assms(1)
  unfolding is-maximal_l-def
  by blast

  then have literal'-grounding: literal.is-ground (literal' ·l γ)
    using assms(2) clause.to-set-is-ground-subst by blast

  have literal-grounding: literal.is-ground (literal ·l γ)
    using assms(1) assms(2) clause.to-set-is-ground-subst by blast

  have literal-γ-in-premise: literal' ·l γ ∈# clause · γ
    using clause.subst-in-to-set-subst[OF literal'(2)]
    by simp

  have literal ·l γ ≺_l literal' ·l γ
      using less_l-ground-subst-stability[OF literal-grounding literal'-grounding literal'(1)].

  then have ¬ is-maximal_l (literal ·l γ) (clause · γ)
    using literal-γ-in-premise
    unfolding is-maximal_l-def literal.subst-comp-subst
    by (metis asympD less_l-asymmetric)

  then show False
    using assms(3)
    by blast
qed

lemma less_l-total-on [intro]: totalp-on (literal.from-ground ` literals_G) (≺_l)
   by (smt (verit, best) image-iff less_lG-less_l totalpD ground.less_lG-total-on totalp-on-def)
```

lemmas *less$_l$-total-on-set-mset* =
less$_l$-total-on[*THEN totalp-on-subset*, *OF clause.to-set-from-ground*[*THEN equalityD1*]]

lemma *less$_c$-total-on*: *totalp-on* (*clause.from-ground* ' *clauses*) ($\prec_c$)
by (*smt ground.clause-order.totalp-on-less image-iff less$_{cG}$-less$_c$ totalpD totalp-onI*)

lemma *unique-maximal-in-ground-clause*:
assumes
clause.is-ground clause
is-maximal$_l$ literal clause
is-maximal$_l$ literal′ clause
shows
literal = *literal′*
using *assms*(*2*, *3*)
unfolding *is-maximal$_l$-def*
by (*metis assms*(*1*) *less$_l$-total-on-set-mset clause.to-ground-inverse totalp-onD*)

lemma *unique-strictly-maximal-in-ground-clause*:
assumes
clause.is-ground clause
is-strictly-maximal$_l$ literal clause
is-strictly-maximal$_l$ literal′ clause
shows
literal = *literal′*
proof−
note *are-maximal$_l$* = *assms*(*2*, *3*)[*THEN is-maximal$_l$-if-is-strictly-maximal$_l$*]

show *?thesis*
using *unique-maximal-in-ground-clause*[*OF assms*(*1*) *are-maximal$_l$*]**.**
qed

lemma *is-strictly-maximal$_l$-ground-subst-stability*:
assumes
clause-grounding: *clause.is-ground* (*clause* · γ) **and**
ground-strictly-maximal: *is-strictly-maximal$_l$ literal$_G$* (*clause* · γ)
obtains *literal* **where**
is-strictly-maximal$_l$ literal clause literal ·*l* γ = *literal$_G$*
proof−
assume *assumption*: $\bigwedge$*literal.*
is-strictly-maximal$_l$ literal clause $\Longrightarrow$ *literal* ·*l* γ = *literal$_G$* $\Longrightarrow$ *thesis*

have *clause-grounding-not-empty*: *clause* · $\gamma \neq$ {#}
using *ground-strictly-maximal*
unfolding *is-strictly-maximal$_l$-def*
by *fastforce*

have *literal$_G$-in-clause-grounding*: *literal$_G$* ∈# *clause* · γ
using *ground-strictly-maximal is-strictly-maximal$_l$-def* **by** *blast*

obtain *literal* **where** *literal*: *literal* $\in\#$ *clause literal* $\cdot l\ \gamma$ = $literal_G$
by (*smt* (*verit*, *best*) *clause.subst-def imageE* $literal_G$*-in-clause-grounding multiset.set-map*)

show *?thesis*
proof(*cases is-strictly-maximal*$_l$ *literal clause*)
case *True*
then show *?thesis*
using *assumption*
using *literal*(*2*) **by** *blast*
next
case *False*

then obtain *literal′* **where** *literal′*:
literal′ $\in\#$ *clause* $-$ {# *literal* #}
literal $\preceq_l$ *literal′*
unfolding *is-strictly-maximal*$_l$*-def*
using *literal*(*1*)
by *blast*

note *literal-grounding* =
clause.to-set-is-ground-subst[*OF literal*(*1*) *clause-grounding*]

have *literal′-grounding*: *literal.is-ground* (*literal′* $\cdot l\ \gamma$)
using *literal′*(*1*) *clause-grounding*
by (*meson clause.to-set-is-ground-subst in-diffD*)

have *literal* $\cdot l\ \gamma \preceq_l$ *literal′* $\cdot l\ \gamma$
using *less-eq*$_l$*-ground-subst-stability*[*OF literal-grounding literal′-grounding literal′*(*2*)].

then have *False*
using *clause.subst-in-to-set-subst*[*OF literal′*(*1*)] *ground-strictly-maximal*
unfolding
is-strictly-maximal$_l$*-def*
literal(*2*)[*symmetric*]
subst-clause-remove1-mset[*OF literal*(*1*)]
by *blast*

then show *?thesis*..
qed
qed

lemma *is-strictly-maximal*$_l$*-ground-subst-stability′*:
assumes
literal $\in\#$ *clause*
clause.is-ground (*clause* $\cdot\ \gamma$)
is-strictly-maximal$_l$ (*literal* $\cdot l\ \gamma$) (*clause* $\cdot\ \gamma$)

shows
  is-strictly-maximal$_l$ *literal clause*
 using
  is-maximal$_l$*-ground-subst-stability'*[*OF*
   assms(*1*,*2*)
   is-maximal$_l$*-if-is-strictly-maximal*$_l$[*OF assms*(*3*)]
  ]
  assms(*3*)
 unfolding
  is-strictly-maximal$_l$*-def is-maximal*$_l$*-def*
  subst-clause-remove1-mset[*OF assms*(*1*), *symmetric*]
 by (*metis in-diffD clause.subst-in-to-set-subst reflclp-iff*)

lemma *less*$_t$*-less*$_l$:
 assumes $term_1 \prec_t term_2$
 shows
  $term_1 \approx term_3 \prec_l term_2 \approx term_3$
  $term_1\ !\!\approx term_3 \prec_l term_2\ !\!\approx term_3$
 using *assms*
 unfolding *less*$_l$*-def multp-eq-multp*$_{HO}$[*OF less*$_t$*-asymmetric less*$_t$*-transitive*] *multp*$_{HO}$*-def*

 by (*auto simp*: *add-mset-eq-add-mset*)

lemma *less*$_t$*-less*$_l$*'*:
 assumes
  $\forall$ *term* $\in$ *set-uprod* (*atm-of literal*). *term* $\cdot t\ \sigma' \preceq_t$ *term* $\cdot t\ \sigma$
  $\exists$ *term* $\in$ *set-uprod* (*atm-of literal*). *term* $\cdot t\ \sigma' \prec_t$ *term* $\cdot t\ \sigma$
 shows *literal* $\cdot l\ \sigma' \prec_l$ *literal* $\cdot l\ \sigma$
proof(*cases literal*)
 case (*Pos atom*)
 show *?thesis*
 proof(*cases atom*)
  case (*Upair* $term_1$ $term_2$)
  have $term_2 \cdot t\ \sigma' \prec_t term_2 \cdot t\ \sigma \Longrightarrow$
     multp $(\prec_t)$ $\{\#term_1 \cdot t\ \sigma,\ term_2 \cdot t\ \sigma'\#\}$ $\{\#term_1 \cdot t\ \sigma,\ term_2 \cdot t\ \sigma\#\}$
    using *multp-add-mset'*[*of* $(\prec_t)$ $term_2 \cdot t\ \sigma'$ $term_2 \cdot t\ \sigma$ $\{\#term_1 \cdot t\ \sigma\#\}$]
add-mset-commute
   by *metis*

  then show *?thesis*
   using *assms*
   unfolding *less*$_l$*-def Pos subst-literal*(*1*) *Upair subst-atom*
   by (*auto simp*: *multp-add-mset multp-add-mset'*)
 qed
next
 case (*Neg atom*)
 show *?thesis*
  proof(*cases atom*)
   case (*Upair* $term_1$ $term_2$)

have $term_2 \cdot t\ \sigma' \prec_t term_2 \cdot t\ \sigma \Longrightarrow$
multp $(\prec_t)$
$\{\#term_1 \cdot t\ \sigma,\ term_1 \cdot t\ \sigma,\ term_2 \cdot t\ \sigma',\ term_2 \cdot t\ \sigma'\#\}$
$\{\#term_1 \cdot t\ \sigma,\ term_1 \cdot t\ \sigma,\ term_2 \cdot t\ \sigma,\ term_2 \cdot t\ \sigma\#\}$
using *multp-add-mset′ multp-add-same*[*OF* $less_t$*-asymmetric* $less_t$*-transitive*]
by *simp*

then show *?thesis*
using *assms*
unfolding $less_l$*-def Neg subst-literal*(*2*) *Upair subst-atom*
by (*auto simp*: *multp-add-mset multp-add-mset′ add-mset-commute*)
qed
qed

lemmas $less_c$*-add-mset* = *multp-add-mset-reflclp*[*OF* $less_l$*-asymmetric* $less_l$*-transitive*, *folded* $less_c$*-def*]

lemmas $less_c$*-add-same* = *multp-add-same*[*OF* $less_l$*-asymmetric* $less_l$*-transitive*, *folded* $less_c$*-def*]

lemma *less-eq*$_l$*-less-eq*$_c$:
assumes $\forall$ *literal* $\in\#$ *clause*. *literal* $\cdot l\ \sigma' \preceq_l$ *literal* $\cdot l\ \sigma$
shows *clause* $\cdot\ \sigma' \preceq_c$ *clause* $\cdot\ \sigma$
using *assms*
by(*induction clause*)(*clause-auto simp*: $less_c$*-add-same* $less_c$*-add-mset*)

lemma $less_l$*-*$less_c$:
assumes
$\forall$ *literal* $\in\#$ *clause*. *literal* $\cdot l\ \sigma' \preceq_l$ *literal* $\cdot l\ \sigma$
$\exists$ *literal* $\in\#$ *clause*. *literal* $\cdot l\ \sigma' \prec_l$ *literal* $\cdot l\ \sigma$
shows *clause* $\cdot\ \sigma' \prec_c$ *clause* $\cdot\ \sigma$
using *assms*
proof(*induction clause*)
case *empty*
then show *?case* **by** *auto*
next
case (*add literal clause*)
then have *less-eq*: $\forall$ *literal* $\in\#$ *clause*. *literal* $\cdot l\ \sigma' \preceq_l$ *literal* $\cdot l\ \sigma$
by (*metis add-mset-remove-trivial in-diffD*)

show *?case*
proof(*cases literal* $\cdot l\ \sigma' \prec_l$ *literal* $\cdot l\ \sigma$)
case *True*
moreover have *clause* $\cdot\ \sigma' \preceq_c$ *clause* $\cdot\ \sigma$
using *less-eq*$_l$*-less-eq*$_c$[*OF less-eq*].

ultimately show *?thesis*
using $less_c$*-add-mset*
unfolding *subst-clause-add-mset* $less_c$*-def*

```
      by blast
  next
    case False
    then have less: ∃ literal ∈# clause. literal ·l σ′ ≺l literal ·l σ
      using add.prems(2) by auto

    from False have eq: literal ·l σ′ = literal ·l σ
      using add.prems(1) by force

    show ?thesis
     using add(1)[OF less-eq less] lessc-add-same
     unfolding subst-clause-add-mset eq lessc-def
     by blast
  qed
qed
```

6.8 Substitution update

```
lemma lesst-subst-upd:
  fixes γ :: ('f, 'v) subst
  assumes
    update-is-ground: term.is-ground update and
    update-less: update ≺t γ var and
    term-grounding: term.is-ground (term ·t γ) and
    var: var ∈ term.vars term
  shows term ·t γ(var := update) ≺t term ·t γ
  using assms(3, 4)
proof(induction term)
  case Var
  then show ?case
    using update-is-ground update-less
    by simp
next
  case (Fun f terms)

  then have ∀ term ∈ set terms. term ·t γ(var := update) ≼t term ·t γ
    by (metis eval-with-fresh-var is-ground-iff reflclp-iff term.set-intros(4))

  moreover then have ∃ term ∈ set terms. term ·t γ(var := update) ≺t term ·t
γ
    using Fun assms(2)
   by (metis (full-types) fun-upd-same  term.distinct(1) term.sel(4) term.set-cases(2)

        term-order.dual-order.strict-iff-order term-subst-eq-rev)

  ultimately show ?case
    using Fun(2, 3)
  proof(induction filter (λterm. term ·t γ(var := update) ≺t term ·t γ) terms
        arbitrary: terms)
```

case *Nil*
then show *?case*
unfolding *empty-filter-conv*
by *blast*
next
case *first*: (*Cons t ts*)

have *update-grounding* [*simp*]: *term.is-ground* (*t* ·*t* γ(*var* := *update*))
using *first.prems*(*3*) *update-is-ground first.hyps*(*2*)
by (*metis* (*no-types*, *lifting*) *filter-eq-ConsD fun-upd-other fun-upd-same in-set-conv-decomp*
is-ground-iff term.set-intros(*4*))

then have *t-grounding* [*simp*]: *term.is-ground* (*t* ·*t* γ)
using *update-grounding Fun.prems*(*1*,*2*)
by (*metis fun-upd-other is-ground-iff*)

show *?case*
proof(*cases ts*)
case *Nil*
then obtain *ss1 ss2* **where** *terms*: *terms* = *ss1* @ *t* # *ss2*
using *filter-eq-ConsD*[*OF first.hyps*(*2*)[*symmetric*]]
by *blast*

have *ss1*: ∀ *term* ∈ *set ss1*. *term* ·*t* γ(*var* := *update*) = *term* ·*t* γ
using *first.hyps*(*2*) *first.prems*(*1*)
unfolding *Nil terms*
by (*smt* (*verit*, *del-insts*) *Un-iff append-Cons-eq-iff filter-empty-conv filter-eq-ConsD*
set-append term-order.antisym-conv2)

have *ss2*: ∀ *term* ∈ *set ss2*. *term* ·*t* γ(*var* := *update*) = *term* ·*t* γ
using *first.hyps*(*2*) *first.prems*(*1*)
unfolding *Nil terms*
by (*smt* (*verit*, *ccfv-SIG*) *Un-iff append-Cons-eq-iff filter-empty-conv filter-eq-ConsD*
list.set-intros(*2*) *set-append term-order.antisym-conv2*)

let *?context* = *More f* (*map* (λ*term*. (*term* ·*t* γ)) *ss1*) □ (*map* (λ*term*. (*term* ·*t* γ)) *ss2*)

have *1*: *term.is-ground* (*t* ·*t* γ)
using *terms first*(*5*)
by *auto*

moreover then have *term.is-ground* (*t* ·*t* γ(*var* := *update*))
by (*metis assms*(*1*) *fun-upd-other fun-upd-same is-ground-iff*)

moreover have *context.is-ground ?context*

```
      using terms first(5)
      by auto

    moreover have t ·t γ(var := update) ≺t t ·t γ
      using first.hyps(2)
      by (meson Cons-eq-filterD)

    ultimately have ?context⟨t ·t γ(var := update)⟩ ≺t ?context⟨t ·t γ⟩
      using lesst-ground-context-compatible
      by blast

    moreover have Fun f terms ·t γ(var := update) = ?context⟨t ·t γ(var :=
update)⟩
      unfolding terms
      using ss1 ss2
      by simp

    moreover have Fun f terms ·t γ = ?context⟨t ·t γ⟩
      unfolding terms
      by auto

    ultimately show ?thesis
      by argo
  next
    case (Cons t' ts')

    from first(2)
    obtain ss1 ss2 where
      terms: terms = ss1 @ t # ss2 and
      ss1: ∀ s∈set ss1. ¬ s ·t γ(var := update) ≺t s ·t γ and
      less: t ·t γ(var := update) ≺t t ·t γ and
      ts: ts = filter (λterm. term ·t γ(var := update)≺t term ·t γ) ss2
      using Cons-eq-filter-iff[of t ts (λterm. term ·t γ(var := update) ≺t term ·t
γ)]
      by blast

    let ?terms' = ss1 @ (t ·t γ(var := update)) # ss2

    have [simp]: t ·t γ(var := update) ·t γ = t ·t γ(var := update)
      using first.prems(3) update-is-ground
      unfolding terms
      by (simp add: is-ground-iff)

    have [simp]: t ·t γ(var := update) ·t γ(var := update) = t ·t γ(var := update)
      using first.prems(3) update-is-ground
      unfolding terms
      by (simp add: is-ground-iff)

    have ts = filter (λterm. term ·t γ(var := update) ≺t term ·t γ) ?terms'
```

using *ss1 ts*
by *auto*

moreover have ∀ *term* ∈ *set ?terms'*. *term* ·*t* γ(*var* := *update*) $\preceq_t$ *term* ·*t* γ
using *first.prems*(*1*)
unfolding *terms*
by *simp*

moreover have ∃ *term* ∈ *set ?terms'*. *term* ·*t* γ(*var* := *update*) $\prec_t$ *term* ·*t* γ
using *calculation*(*1*) *Cons neq-Nil-conv* **by** *force*

moreover have *terms'-grounding*: *term.is-ground* (*Fun f ?terms'* ·*t* γ)
using *first.prems*(*3*)
unfolding *terms*
by *simp*

moreover have *var* ∈ *term.vars* (*Fun f ?terms'*)
by (*metis calculation*(*3*) *eval-with-fresh-var term.set-intros*(*4*) *term-order.less-irrefl*)

ultimately have *less-terms'*: *Fun f ?terms'* ·*t* γ(*var* := *update*) $\prec_t$ *Fun f ?terms'* ·*t* γ
using *first.hyps*(*1*) *first.prems*(*3*) **by** *blast*

have *context-grounding*: *context.is-ground* (*More f ss1* □ *ss2* ·t_c γ)
using *terms'-grounding*
by *auto*

have *Fun f* (*ss1* @ *t* ·*t* γ(*var* := *update*) # *ss2*) ·*t* γ $\prec_t$ *Fun f terms* ·*t* γ
unfolding *terms*
using $less_t$*-ground-context-compatible*[*OF less - - context-grounding*]
by *simp*

with *less-terms'* **show** *?thesis*
unfolding *terms*
by *auto*
qed
qed
qed

lemma $less_l$*-subst-upd*:
fixes γ :: (*'f*, *'v*) *subst*
assumes
update-is-ground: *term.is-ground update* **and**
update-less: *update* $\prec_t$ γ *var* **and**
literal-grounding: *literal.is-ground* (*literal* ·*l* γ) **and**
var: *var* ∈ *literal.vars literal*
shows *literal* ·*l* γ(*var* := *update*) $\prec_l$ *literal* ·*l* γ
proof−
note $less_t$*-subst-upd* = $less_t$*-subst-upd*[*of - γ*, *OF update-is-ground update-less*]

have *all-ground-terms*: ∀ *term* ∈ *set-uprod* (*atm-of literal*). *term.is-ground* (*term* ·*t* γ)
  using *assms*(*3*)
  apply(*cases literal*)
  by (*simp add*: *ground-term-in-ground-literal-subst*)+

then have
  ∀ *term* ∈ *set-uprod* (*atm-of literal*).
    var ∈ *term.vars term* ⟶ *term* ·*t* γ(*var* := *update*) $\prec_t$ *term* ·*t* γ
  using $less_t$*-subst-upd*
  by *blast*

moreover have
  ∀ *term* ∈ *set-uprod* (*atm-of literal*).
    var ∉ *term.vars term* ⟶ *term* ·*t* γ(*var* := *update*) = *term* ·*t* γ
  by (*meson eval-with-fresh-var*)

ultimately have ∀ *term* ∈ *set-uprod* (*atm-of literal*). *term* ·*t* γ(*var* := *update*) $\preceq_t$ *term* ·*t* γ
  by *blast*

moreover have ∃ *term* ∈ *set-uprod* (*atm-of literal*). *term* ·*t* γ(*var* := *update*) $\prec_t$ *term* ·*t* γ
  using *update-less var* $less_t$*-subst-upd all-ground-terms*
  unfolding *literal.vars-def atom.vars-def set-literal-atm-of*
  by *blast*

ultimately show *?thesis*
  using $less_t$-$less_l$′
  by *blast*
qed

lemma $less_c$*-subst-upd*:
  assumes
    update-is-ground: *term.is-ground update* **and**
    update-less: *update* $\prec_t$ γ *var* **and**
    literal-grounding: *clause.is-ground* (*clause* · γ) **and**
    var: *var* ∈ *clause.vars clause*
  shows *clause* · γ(*var* := *update*) $\prec_c$ *clause* · γ
proof−
  note $less_l$*-subst-upd* = $less_l$*-subst-upd*[*of* - γ, *OF update-is-ground update-less*]

have *all-ground-literals*: ∀ *literal* ∈# *clause*. *literal.is-ground* (*literal* ·*l* γ)
  using *clause.to-set-is-ground-subst*[*OF* - *literal-grounding*] **by** *blast*

then have
  ∀ *literal* ∈# *clause*.
    var ∈ *literal.vars literal* ⟶ *literal* ·*l* γ(*var* := *update*) $\prec_l$ *literal* ·*l* γ

using *less*$_l$*-subst-upd*
by *blast*

then have $\forall$ *literal* $\in\#$ *clause. literal* $\cdot l$ $\gamma(var := update) \preceq_l$ *literal* $\cdot l$ γ
by *fastforce*

moreover have $\exists$ *literal* $\in\#$ *clause. literal* $\cdot l$ $\gamma(var := update) \prec_l$ *literal* $\cdot l$ γ
using *update-less var less*$_l$*-subst-upd all-ground-literals*
unfolding *clause.vars-def*
by *blast*

ultimately show *?thesis*
using *less*$_l$*-less*$_c$
by *blast*
qed

end

end
theory *First-Order-Superposition*
imports
Saturation-Framework.Lifting-to-Non-Ground-Calculi
Ground-Superposition
First-Order-Select
First-Order-Ordering
First-Order-Type-System
begin

hide-type *Inference-System.inference*
hide-const
Inference-System.Infer
Inference-System.prems-of
Inference-System.concl-of
Inference-System.main-prem-of

hide-fact
Restricted-Predicates.wfp-on-imp-minimal
Restricted-Predicates.wfp-on-imp-inductive-on
Restricted-Predicates.inductive-on-imp-wfp-on
Restricted-Predicates.wfp-on-iff-inductive-on
Restricted-Predicates.wfp-on-iff-minimal
Restricted-Predicates.wfp-on-imp-has-min-elt
Restricted-Predicates.wfp-on-induct
Restricted-Predicates.wfp-on-UNIV
Restricted-Predicates.wfp-less
Restricted-Predicates.wfp-on-measure-on
Restricted-Predicates.wfp-on-mono
Restricted-Predicates.wfp-on-subset

Restricted-Predicates.wfp-on-restrict-to
Restricted-Predicates.wfp-on-imp-irreflp-on
Restricted-Predicates.accessible-on-imp-wfp-on
Restricted-Predicates.wfp-on-tranclp-imp-wfp-on
Restricted-Predicates.wfp-on-imp-accessible-on
Restricted-Predicates.wfp-on-accessible-on-iff
Restricted-Predicates.wfp-on-restrict-to-tranclp
Restricted-Predicates.wfp-on-restrict-to-tranclp'
Restricted-Predicates.wfp-on-restrict-to-tranclp-wfp-on-conv

7 First-Order Layer

locale *first-order-superposition-calculus* =
first-order-select select +
first-order-ordering $less_t$
for
select :: ('*f*, ('*v* :: *infinite*)) *select* **and**
$less_t$ :: ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *term* ⇒ *bool* (**infix** $\prec_t$ *50*) +
fixes
tiebreakers :: '*f gatom clause* ⇒ ('*f*, '*v*) *atom clause* ⇒ ('*f*, '*v*) *atom clause* ⇒ *bool* **and**
typeof-fun :: ('*f*, '*ty*) *fun-types*
assumes
wellfounded-tiebreakers:
$\bigwedge clause_G$. *wfP* (*tiebreakers* $clause_G$) ∧
transp (*tiebreakers* $clause_G$) ∧
asymp (*tiebreakers* $clause_G$) **and**
function-symbols: $\bigwedge\tau$. $\exists f$. *typeof-fun f* = ([], τ) **and**
ground-critical-pair-theorem: $\bigwedge$(R :: '*f gterm rel*). *ground-critical-pair-theorem* R **and**
variables: |*UNIV* :: '*ty set*| $\leq o$ |*UNIV* :: '*v set*|
begin

abbreviation *typed-tiebreakers* ::
'*f gatom clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ ('*f*, '*v*, '*ty*) *typed-clause* ⇒ *bool*
where
typed-tiebreakers $clause_G$ $clause_1$ $clause_2$ ≡ *tiebreakers* $clause_G$ (*fst* $clause_1$) (*fst* $clause_2$)

lemma *wellfounded-typed-tiebreakers*:
wfP (*typed-tiebreakers* $clause_G$) ∧
transp (*typed-tiebreakers* $clause_G$) ∧
asymp (*typed-tiebreakers* $clause_G$)
proof(*intro conjI*)

show *wfp* (*typed-tiebreakers* $clause_G$)
using *wellfounded-tiebreakers*
by (*meson wfP-if-convertible-to-wfP*)

show *transp* (*typed-tiebreakers* $clause_G$)
using *wellfounded-tiebreakers*
by (*smt* (*verit*, *ccfv-threshold*) *transpD transpI*)

show *asymp* (*typed-tiebreakers* $clause_G$)
using *wellfounded-tiebreakers*
by (*meson asympD asympI*)
qed

definition *is-merged-var-type-env* **where**
is-merged-var-type-env $\mathcal{V}$ X $\mathcal{V}_X$ ϱ_X Y $\mathcal{V}_Y$ ϱ_Y $\equiv$
$(\forall x \in X.$ *welltyped typeof-fun* $\mathcal{V}$ $(\varrho_X\ x)$ $(\mathcal{V}_X\ x)) \wedge$
$(\forall y \in Y.$ *welltyped typeof-fun* $\mathcal{V}$ $(\varrho_Y\ y)$ $(\mathcal{V}_Y\ y))$

inductive *eq-resolution* :: $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ *bool* **where**
eq-resolutionI:
premise = *add-mset literal premise'* $\Longrightarrow$
literal = *term* !$\approx$ *term'* $\Longrightarrow$
term-subst.is-imgu μ {{ *term*, *term'* }} $\Longrightarrow$
welltyped-imgu' *typeof-fun* $\mathcal{V}$ *term term'* μ $\Longrightarrow$
select premise = {#} $\wedge$ $is\text{-}maximal_l$ (*literal* $\cdot l$ μ) (*premise* $\cdot$ μ) $\vee$
$is\text{-}maximal_l$ (*literal* $\cdot l$ μ) ((*select premise*) $\cdot$ μ) $\Longrightarrow$
conclusion = *premise'* $\cdot$ μ $\Longrightarrow$
eq-resolution (*premise*, $\mathcal{V}$) (*conclusion*, $\mathcal{V}$)

inductive *eq-factoring* :: $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ *bool* **where**
eq-factoringI:
premise = *add-mset* $literal_1$ (*add-mset* $literal_2$ *premise'*) $\Longrightarrow$
$literal_1 = term_1 \approx term_1{}'$ $\Longrightarrow$
$literal_2 = term_2 \approx term_2{}'$ $\Longrightarrow$
select premise = {#} $\Longrightarrow$
$is\text{-}maximal_l$ ($literal_1$ $\cdot l$ μ) (*premise* $\cdot$ μ) $\Longrightarrow$
$\neg$ ($term_1 \cdot t\ \mu \preceq_t term_1{}' \cdot t\ \mu$) $\Longrightarrow$
term-subst.is-imgu μ {{ $term_1$, $term_2$ }} $\Longrightarrow$
welltyped-imgu' *typeof-fun* $\mathcal{V}$ $term_1$ $term_2$ μ $\Longrightarrow$
conclusion = *add-mset* ($term_1 \approx term_2{}'$) (*add-mset* ($term_1{}'$!$\approx$ $term_2{}'$) *premise'*) $\cdot$ μ $\Longrightarrow$
eq-factoring (*premise*, $\mathcal{V}$) (*conclusion*, $\mathcal{V}$)

inductive *superposition* ::
$('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ *bool*
where
superpositionI:
term-subst.is-renaming ϱ_1 $\Longrightarrow$
term-subst.is-renaming ϱ_2 $\Longrightarrow$
clause.vars ($premise_1$ $\cdot$ ϱ_1) $\cap$ *clause.vars* ($premise_2$ $\cdot$ ϱ_2) = {} $\Longrightarrow$

$premise_1 = add\text{-}mset\ literal_1\ premise_1{}' \Longrightarrow$
$premise_2 = add\text{-}mset\ literal_2\ premise_2{}' \Longrightarrow$
$\mathcal{P} \in \{Pos, Neg\} \Longrightarrow$
$literal_1 = \mathcal{P}\ (Upair\ context_1\langle term_1\rangle\ term_1{}') \Longrightarrow$
$literal_2 = term_2 \approx term_2{}' \Longrightarrow$
$\neg\ is\text{-}Var\ term_1 \Longrightarrow$
$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{term_1 \cdot t\ \varrho_1,\ term_2 \cdot t\ \varrho_2\}\} \Longrightarrow$
$welltyped\text{-}imgu'\ typeof\text{-}fun\ \mathcal{V}_3\ (term_1 \cdot t\ \varrho_1)\ (term_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in clause.vars\ (premise_1 \cdot \varrho_1).\ \mathcal{V}_1\ (the\text{-}inv\ \varrho_1\ (Var\ x)) = \mathcal{V}_3\ x \Longrightarrow$
$\forall x \in clause.vars\ (premise_2 \cdot \varrho_2).\ \mathcal{V}_2\ (the\text{-}inv\ \varrho_2\ (Var\ x)) = \mathcal{V}_3\ x \Longrightarrow$
$welltyped_\sigma\text{-}on\ (clause.vars\ premise_1)\ typeof\text{-}fun\ \mathcal{V}_1\ \varrho_1 \Longrightarrow$
$welltyped_\sigma\text{-}on\ (clause.vars\ premise_2)\ typeof\text{-}fun\ \mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge\tau\ \tau'.\ has\text{-}type\ typeof\text{-}fun\ \mathcal{V}_2\ term_2\ \tau \Longrightarrow has\text{-}type\ typeof\text{-}fun\ \mathcal{V}_2\ term_2{}'\ \tau'$
$\Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (premise_1 \cdot \varrho_1 \cdot \mu \preceq_c premise_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
$(\mathcal{P} = Pos$
$\wedge\ select\ premise_1 = \{\#\}$
$\wedge\ is\text{-}strictly\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ (premise_1 \cdot \varrho_1 \cdot \mu)) \vee$
$(\mathcal{P} = Neg$
$\wedge\ (select\ premise_1 = \{\#\} \wedge is\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ (premise_1 \cdot \varrho_1 \cdot$
$\mu)$
$\vee\ is\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ ((select\ premise_1) \cdot \varrho_1 \cdot \mu))) \Longrightarrow$
$select\ premise_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal_l\ (literal_2 \cdot l\ \varrho_2 \cdot l\ \mu)\ (premise_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
$\neg\ (context_1\langle term_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \preceq_t term_1{}' \cdot t\ \varrho_1 \cdot t\ \mu) \Longrightarrow$
$\neg\ (term_2 \cdot t\ \varrho_2 \cdot t\ \mu \preceq_t term_2{}' \cdot t\ \varrho_2 \cdot t\ \mu) \Longrightarrow$
$conclusion = add\text{-}mset\ (\mathcal{P}\ (Upair\ (context_1 \cdot t_c\ \varrho_1)\langle term_2{}' \cdot t\ \varrho_2\rangle\ (term_1{}' \cdot t\ \varrho_1)))$

$(premise_1{}' \cdot \varrho_1 + premise_2{}' \cdot \varrho_2) \cdot \mu \Longrightarrow$
$all\text{-}types\ \mathcal{V}_1 \Longrightarrow all\text{-}types\ \mathcal{V}_2 \Longrightarrow$
$superposition\ (premise_2, \mathcal{V}_2)\ (premise_1, \mathcal{V}_1)\ (conclusion, \mathcal{V}_3)$

abbreviation *eq-factoring-inferences* **where**
eq-factoring-inferences ≡
{ *Infer* [*premise*] *conclusion* | *premise conclusion. eq-factoring premise conclusion*
}

abbreviation *eq-resolution-inferences* **where**
eq-resolution-inferences ≡
{ *Infer* [*premise*] *conclusion* | *premise conclusion. eq-resolution premise conclusion* }

abbreviation *superposition-inferences* **where**
$superposition\text{-}inferences \equiv \{\ Infer\ [premise_2,\ premise_1]\ conclusion$
$|\ \ premise_2\ premise_1\ conclusion.\ superposition\ premise_2\ premise_1\ conclusion\}$

definition *inferences* :: ('*f*, '*v*, '*ty*) *typed-clause inference set* **where**
inferences ≡ *superposition-inferences* ∪ *eq-resolution-inferences* ∪ *eq-factoring-inferences*

abbreviation *bottom*$_F$ :: ('*f*, '*v*, '*ty*) *typed-clause set* ($\bot_F$) **where**
bottom$_F$ $\equiv$ {({#}, $\mathcal{V}$) | $\mathcal{V}$. *all-types* $\mathcal{V}$ }

7.0.1 Alternative Specification of the Superposition Rule

inductive *pos-superposition* ::
('*f*, '*v*, '*ty*) *typed-clause* $\Rightarrow$ ('*f*, '*v*, '*ty*) *typed-clause* $\Rightarrow$ ('*f*, '*v*, '*ty*) *typed-clause* $\Rightarrow$ *bool*
where
pos-superpositionI:
term-subst.is-renaming ϱ_1 $\Longrightarrow$
term-subst.is-renaming ϱ_2 $\Longrightarrow$
clause.vars ($P_1 \cdot \varrho_1$) $\cap$ *clause.vars* ($P_2 \cdot \varrho_2$) = {} $\Longrightarrow$
P_1 = *add-mset* L_1 $P_1{}'$ $\Longrightarrow$
P_2 = *add-mset* L_2 $P_2{}'$ $\Longrightarrow$
$L_1 = s_1\langle u_1\rangle \approx s_1{}'$ $\Longrightarrow$
$L_2 = t_2 \approx t_2{}'$ $\Longrightarrow$
$\neg$ *is-Var* u_1 $\Longrightarrow$
term-subst.is-imgu μ {{$u_1 \cdot t\ \varrho_1$, $t_2 \cdot t\ \varrho_2$}} $\Longrightarrow$
welltyped-imgu' *typeof-fun* $\mathcal{V}_3$ ($u_1 \cdot t\ \varrho_1$) ($t_2 \cdot t\ \varrho_2$) μ $\Longrightarrow$
$\forall\, x \in$ *clause.vars* ($P_1 \cdot \varrho_1$). $\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var* x)) = $\mathcal{V}_3$ x $\Longrightarrow$
$\forall\, x \in$ *clause.vars* ($P_2 \cdot \varrho_2$). $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var* x)) = $\mathcal{V}_3$ x $\Longrightarrow$
welltyped$_\sigma$*-on* (*clause.vars* P_1) *typeof-fun* $\mathcal{V}_1$ ϱ_1 $\Longrightarrow$
welltyped$_\sigma$*-on* (*clause.vars* P_2) *typeof-fun* $\mathcal{V}_2$ ϱ_2 $\Longrightarrow$
($\bigwedge\tau\ \tau'$. *has-type typeof-fun* $\mathcal{V}_2$ t_2 τ $\Longrightarrow$ *has-type typeof-fun* $\mathcal{V}_2$ $t_2{}'$ τ' $\Longrightarrow$ $\tau = \tau'$) $\Longrightarrow$
$\neg$ ($P_1 \cdot \varrho_1 \cdot \mu \preceq_c P_2 \cdot \varrho_2 \cdot \mu$) $\Longrightarrow$
select P_1 = {#} $\Longrightarrow$
is-strictly-maximal$_l$ ($L_1 \cdot l\ \varrho_1 \cdot l\ \mu$) ($P_1 \cdot \varrho_1 \cdot \mu$) $\Longrightarrow$
select P_2 = {#} $\Longrightarrow$
is-strictly-maximal$_l$ ($L_2 \cdot l\ \varrho_2 \cdot l\ \mu$) ($P_2 \cdot \varrho_2 \cdot \mu$) $\Longrightarrow$
$\neg$ ($s_1\langle u_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \preceq_t s_1{}' \cdot t\ \varrho_1 \cdot t\ \mu$) $\Longrightarrow$
$\neg$ ($t_2 \cdot t\ \varrho_2 \cdot t\ \mu \preceq_t t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu$) $\Longrightarrow$
C = *add-mset* (($s_1 \cdot t_c\ \varrho_1$)$\langle t_2{}' \cdot t\ \varrho_2\rangle \approx$ ($s_1{}' \cdot t\ \varrho_1$)) ($P_1{}' \cdot \varrho_1 + P_2{}' \cdot \varrho_2$) $\cdot\ \mu$ $\Longrightarrow$
all-types $\mathcal{V}_1$ $\Longrightarrow$ *all-types* $\mathcal{V}_2$ $\Longrightarrow$
pos-superposition (P_2, $\mathcal{V}_2$) (P_1, $\mathcal{V}_1$) (C, $\mathcal{V}_3$)

lemma *superposition-if-pos-superposition*:
assumes *pos-superposition* P_2 P_1 C
shows *superposition* P_2 P_1 C
using *assms*
proof (*cases rule*: *pos-superposition.cases*)
case (*pos-superpositionI* ϱ_1 ϱ_2 P_1 P_2 L_1 $P_1{}'$ L_2 $P_2{}'$ s_1 u_1 $s_1{}'$ t_2 $t_2{}'$ μ $\mathcal{V}_3$ $\mathcal{V}_1$ $\mathcal{V}_2$ C)
then show *?thesis*
using *superpositionI*[*of* ϱ_1 ϱ_2 P_1 P_2]
by *blast*
qed

inductive *neg-superposition* ::
$('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ $('f, 'v, 'ty)$ *typed-clause* $\Rightarrow$ *bool*
where
neg-superpositionI:
term-subst.is-renaming $\varrho_1 \Longrightarrow$
term-subst.is-renaming $\varrho_2 \Longrightarrow$
clause.vars $(P_1 \cdot \varrho_1) \cap$ *clause.vars* $(P_2 \cdot \varrho_2) = \{\} \Longrightarrow$
$P_1 =$ *add-mset* L_1 $P_1' \Longrightarrow$
$P_2 =$ *add-mset* L_2 $P_2' \Longrightarrow$
$L_1 = s_1\langle u_1\rangle$!$\approx$ $s_1' \Longrightarrow$
$L_2 = t_2 \approx t_2' \Longrightarrow$
$\neg$ *is-Var* $u_1 \Longrightarrow$
term-subst.is-imgu μ $\{\{u_1 \cdot t\ \varrho_1,\ t_2 \cdot t\ \varrho_2\}\} \Longrightarrow$
welltyped-imgu' *typeof-fun* $\mathcal{V}_3$ $(u_1 \cdot t\ \varrho_1)$ $(t_2 \cdot t\ \varrho_2)$ $\mu \Longrightarrow$
$\forall\, x \in$ *clause.vars* $(P_1 \cdot \varrho_1)$. $\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var* x)) $= \mathcal{V}_3\ x \Longrightarrow$
$\forall\, x \in$ *clause.vars* $(P_2 \cdot \varrho_2)$. $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var* x)) $= \mathcal{V}_3\ x \Longrightarrow$
welltyped$_\sigma$*-on* (*clause.vars* P_1) *typeof-fun* $\mathcal{V}_1$ $\varrho_1 \Longrightarrow$
welltyped$_\sigma$*-on* (*clause.vars* P_2) *typeof-fun* $\mathcal{V}_2$ $\varrho_2 \Longrightarrow$
$(\bigwedge\tau\ \tau'.$ *has-type* *typeof-fun* $\mathcal{V}_2$ t_2 $\tau \Longrightarrow$ *has-type* *typeof-fun* $\mathcal{V}_2$ t_2' $\tau' \Longrightarrow \tau = \tau') \Longrightarrow$
$\neg$ $(P_1 \cdot \varrho_1 \cdot \mu \preceq_c P_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
select $P_1 = \{\#\} \wedge$
is-maximal$_l$ $(L_1 \cdot l\ \varrho_1 \cdot l\ \mu)$ $(P_1 \cdot \varrho_1 \cdot \mu) \vee$ *is-maximal*$_l$ $(L_1 \cdot l\ \varrho_1 \cdot l\ \mu)$ ((*select* P_1) $\cdot\ \varrho_1 \cdot \mu) \Longrightarrow$
select $P_2 = \{\#\} \Longrightarrow$
is-strictly-maximal$_l$ $(L_2 \cdot l\ \varrho_2 \cdot l\ \mu)$ $(P_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
$\neg$ $(s_1\langle u_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \preceq_t s_1' \cdot t\ \varrho_1 \cdot t\ \mu) \Longrightarrow$
$\neg$ $(t_2 \cdot t\ \varrho_2 \cdot t\ \mu \preceq_t t_2' \cdot t\ \varrho_2 \cdot t\ \mu) \Longrightarrow$
$C =$ *add-mset* (*Neg* (*Upair* $(s_1 \cdot t_c\ \varrho_1)\langle t_2' \cdot t\ \varrho_2\rangle$ $(s_1' \cdot t\ \varrho_1)$))) $(P_1' \cdot \varrho_1 + P_2' \cdot \varrho_2) \cdot \mu \Longrightarrow$
all-types $\mathcal{V}_1 \Longrightarrow$ *all-types* $\mathcal{V}_2 \Longrightarrow$
neg-superposition $(P_2, \mathcal{V}_2)$ $(P_1, \mathcal{V}_1)$ $(C, \mathcal{V}_3)$

lemma *superposition-if-neg-superposition*:
assumes *neg-superposition* P_2 P_1 C
shows *superposition* P_2 P_1 C
using *assms*
proof (*cases* P_2 P_1 C *rule*: *neg-superposition.cases*)
case (*neg-superpositionI* ϱ_1 ϱ_2 P_1 L_1 P_1' P_2 L_2 P_2' s_1 u_1 s_1' t_2 t_2' μ $\mathcal{V}_3$ $\mathcal{V}_1$ $\mathcal{V}_2$ C)
then show *?thesis*
using *superpositionI*[*of* ϱ_1 ϱ_2 P_1 L_1 P_1' P_2 L_2 P_2']
by *blast*
qed

lemma *superposition-iff-pos-or-neg*:
superposition P_2 P_1 $C \longleftrightarrow$ *pos-superposition* P_2 P_1 $C \vee$ *neg-superposition* P_2 P_1 C

proof (*rule iffI*)
assume *superposition* P_2 P_1 C
thus *pos-superposition* P_2 P_1 C $\vee$ *neg-superposition* P_2 P_1 C
proof (*cases* P_2 P_1 C *rule*: *superposition.cases*)
case (*superpositionI* ϱ_1 ϱ_2 $premise_1$ $premise_2$ $literal_1$ $premise_1'$ $literal_2$ $premise_2'$ $\mathcal{P}$ $context_1$
$term_1$ $term_1'$ $term_2$ $term_2'$ μ)
then show *?thesis*
using
pos-superpositionI[*of* ϱ_1 ϱ_2 $premise_1$ $premise_2$ $literal_1$ $premise_1'$ $literal_2$ $premise_2'$ $context_1$
$term_1$ $term_1'$ $term_2$ $term_2'$ μ]
neg-superpositionI[*of* ϱ_1 ϱ_2 $premise_1$ $premise_2$ $literal_1$ $premise_1'$ $literal_2$ $premise_2'$ $context_1$
$term_1$ $term_1'$ $term_2$ $term_2'$ μ]
by *blast*
qed
next
assume *pos-superposition* P_2 P_1 C $\vee$ *neg-superposition* P_2 P_1 C
thus *superposition* P_2 P_1 C
using *superposition-if-neg-superposition superposition-if-pos-superposition* **by** *metis*
qed

lemma *eq-resolution-preserves-typing*:
assumes
step: *eq-resolution* $(D, \mathcal{V})$ $(C, \mathcal{V})$ **and**
wt-D: $welltyped_c$ *typeof-fun* $\mathcal{V}$ D
shows $welltyped_c$ *typeof-fun* $\mathcal{V}$ C
using *step*
proof (*cases* $(D, \mathcal{V})$ $(C, \mathcal{V})$ *rule*: *eq-resolution.cases*)
case (*eq-resolutionI literal premise' term term'* μ)
obtain τ **where** τ:
welltyped typeof-fun $\mathcal{V}$ *term* τ
welltyped typeof-fun $\mathcal{V}$ *term'* τ
using *wt-D*
unfolding
eq-resolutionI
$welltyped_c$*-add-mset*
$welltyped_l$*-def*
$welltyped_a$*-def*
by *clause-simp*

then have $welltyped_c$ *typeof-fun* $\mathcal{V}$ $(D \cdot \mu)$
using *wt-D* $welltyped_\sigma$-$welltyped_c$ *eq-resolutionI*(*4*)
by *blast*

then show *?thesis*
unfolding *eq-resolutionI subst-clause-add-mset* $welltyped_c$*-add-mset*

by *clause-simp*
qed

lemma *has-type-welltyped*:
assumes *has-type typeof-fun* $\mathcal{V}$ *term* τ *welltyped typeof-fun* $\mathcal{V}$ *term* τ'
shows *welltyped typeof-fun* $\mathcal{V}$ *term* τ
using *assms*
by (*smt* (*verit*, *best*) *welltyped.simps has-type.simps has-type-right-unique right-uniqueD*)

lemma *welltyped-has-type*:
assumes *welltyped typeof-fun* $\mathcal{V}$ *term* τ
shows *has-type typeof-fun* $\mathcal{V}$ *term* τ
using *assms welltyped.cases has-type.simps* **by** *fastforce*

lemma *eq-factoring-preserves-typing*:
assumes
step: *eq-factoring* (D, $\mathcal{V}$) (C, $\mathcal{V}$) **and**
wt-D: $welltyped_c$ *typeof-fun* $\mathcal{V}$ D
shows $welltyped_c$ *typeof-fun* $\mathcal{V}$ C
using *step*
proof (*cases* (D, $\mathcal{V}$) (C, $\mathcal{V}$) *rule*: *eq-factoring.cases*)
case (*eq-factoringI* $literal_1$ $literal_2$ *premise'* $term_1$ $term_1'$ $term_2$ $term_2'$ μ)

have *wt-D*μ: $welltyped_c$ *typeof-fun* $\mathcal{V}$ ($D \cdot \mu$)
using *wt-D* $welltyped_\sigma$-$welltyped_c$ *eq-factoringI*
by *blast*

show *?thesis*
proof−
have $\bigwedge\tau\ \tau'$.
⟦∀ L∈#*premise'* · μ.
∃ τ. ∀ t∈*set-uprod* (*atm-of* L). *First-Order-Type-System.welltyped typeof-fun*
$\mathcal{V}$ t τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_1$ ·t μ) τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_1'$ ·t μ) τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_2$ ·t μ) τ';
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_2'$ ·t μ) τ'⟧
⟹ ∃ τ. *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}$ ($term_1$ ·t μ) τ ∧
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_2'$ ·t μ) τ
by (*metis welltyped-right-unique eq-factoringI*(*8*) *right-uniqueD* $welltyped_\sigma$-*welltyped*)

moreover have $\bigwedge\tau\ \tau'$.
⟦∀ L∈#*premise'* · μ.
∃ τ. ∀ t∈*set-uprod* (*atm-of* L). *First-Order-Type-System.welltyped typeof-fun*
$\mathcal{V}$ t τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_1$ ·t μ) τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_1'$ ·t μ) τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_2$ ·t μ) τ';
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ ($term_2'$ ·t μ) τ'⟧

$\Longrightarrow \exists\,\tau.$ *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}$ (*term*$_1$' $\cdot t$ μ) τ $\wedge$
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}$ (*term*$_2$' $\cdot t$ μ) τ
by (*metis welltyped-right-unique eq-factoringI*(*8*) *right-uniqueD welltyped*$_\sigma$*-welltyped*)

ultimately show *?thesis*
using *wt-D*μ
unfolding *welltyped*$_c$*-def welltyped*$_l$*-def welltyped*$_a$*-def eq-factoringI subst-clause-add-mset*

subst-literal subst-atom
by *auto*
qed
qed

lemma *superposition-preserves-typing*:
assumes
step: *superposition* (D, $\mathcal{V}_2$) (C, $\mathcal{V}_1$) (E, $\mathcal{V}_3$) **and**
wt-C: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_1$ C **and**
wt-D: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_2$ D
shows *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ E
using *step*
proof (*cases* (D, $\mathcal{V}_2$) (C, $\mathcal{V}_1$) (E, $\mathcal{V}_3$) *rule*: *superposition.cases*)
case (*superpositionI* ϱ_1 ϱ_2 *literal*$_1$ *premise*$_1$' *literal*$_2$ *premise*$_2$' $\mathcal{P}$ *context*$_1$ *term*$_1$
term$_1$' *term*$_2$
term$_2$' μ)

have *welltyped-*μ: *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}_3$ μ
using *superpositionI*(*11*)
by *blast*

have *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ ($C \cdot \varrho_1$)
using *wt-C welltyped*$_c$*-renaming-weaker*[*OF superpositionI*(*1*, *12*)]
by *blast*

then have *wt-C*μ: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ ($C \cdot \varrho_1 \cdot \mu$)
using *welltyped*$_\sigma$*-welltyped*$_c$[*OF welltyped-*μ]
by *blast*

have *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ ($D \cdot \varrho_2$)
using *wt-D welltyped*$_c$*-renaming-weaker*[*OF superpositionI*(*2*, *13*)]
by *blast*

then have *wt-D*μ: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ ($D \cdot \varrho_2 \cdot \mu$)
using *welltyped*$_\sigma$*-welltyped*$_c$[*OF welltyped-*μ]
by *blast*

note *imgu* = *term-subst.subst-imgu-eq-subst-imgu*[*OF superpositionI*(*10*)]

show *?thesis*
using *literal-cases*[*OF superpositionI*(*6*)] *wt-C*μ *wt-D*μ

by *cases* (*clause-simp simp*: *superpositionI imgu*)
qed

end

end
theory *Grounded-First-Order-Superposition*
imports
First-Order-Superposition
Ground-Superposition-Completeness
begin

context *ground-superposition-calculus*
begin

abbreviation *eq-resolution-inferences* **where**
eq-resolution-inferences ≡ {*Infer* [*P*] *C* | *P C*. *ground-eq-resolution P C*}

abbreviation *eq-factoring-inferences* **where**
eq-factoring-inferences ≡ {*Infer* [*P*] *C* | *P C*. *ground-eq-factoring P C*}

abbreviation *superposition-inferences* **where**
superposition-inferences ≡ {*Infer* [*P2*, *P1*] *C* | *P1 P2 C*. *ground-superposition P2 P1 C*}

end

locale *grounded-first-order-superposition-calculus* =
first-order-superposition-calculus select - - typeof-fun +
grounded-first-order-select select
for
select :: (′*f*, ′*v* :: *infinite*) *select* **and**
typeof-fun :: (′*f*, ′*ty*) *fun-types*
begin

sublocale *ground*: *ground-superposition-calculus* **where**
less-trm = ($\prec_{tG}$) **and** *select* = $select_G$
by *unfold-locales* (*rule ground-critical-pair-theorem*)

definition *is-inference-grounding* **where**
is-inference-grounding ι ι_G γ ϱ_1 ϱ_2 ≡
(*case* ι *of*
Infer [(*premise*, $\mathcal{V}'$)] (*conclusion*, $\mathcal{V}$) ⇒
term-subst.is-ground-subst γ
∧ ι_G = *Infer* [*clause.to-ground* (*premise* · γ)] (*clause.to-ground* (*conclusion* · γ))
∧ $welltyped_c$ *typeof-fun* $\mathcal{V}$ *premise*
∧ $welltyped_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}$ γ
∧ $welltyped_c$ *typeof-fun* $\mathcal{V}$ *conclusion*

$\wedge\ \mathcal{V} = \mathcal{V}'$
$\wedge$ *all-types* $\mathcal{V}$
| *Infer* [(*premise*$_2$, $\mathcal{V}_2$), (*premise*$_1$, $\mathcal{V}_1$)] (*conclusion*, $\mathcal{V}_3$) $\Rightarrow$
term-subst.is-renaming ϱ_1
$\wedge$ *term-subst.is-renaming* ϱ_2
$\wedge$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\cap$ *clause.vars* (*premise*$_2$ $\cdot$ ϱ_2) = {}
$\wedge$ *term-subst.is-ground-subst* γ
$\wedge$ ι_G =
Infer
[*clause.to-ground* (*premise*$_2$ $\cdot$ ϱ_2 $\cdot$ γ), *clause.to-ground* (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ γ)]
(*clause.to-ground* (*conclusion* $\cdot$ γ))
$\wedge$ *welltyped*$_c$ *typeof-fun* $\mathcal{V}_1$ *premise*$_1$
$\wedge$ *welltyped*$_c$ *typeof-fun* $\mathcal{V}_2$ *premise*$_2$
$\wedge$ *welltyped*$_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}_3$ γ
$\wedge$ *welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ *conclusion*
$\wedge$ *all-types* $\mathcal{V}_1$ $\wedge$ *all-types* $\mathcal{V}_2$ $\wedge$ *all-types* $\mathcal{V}_3$
| - $\Rightarrow$ *False*
)
$\wedge$ $\iota_G \in$ *ground.G-Inf*

definition *inference-groundings* **where**
inference-groundings ι = { ι_G | ι_G γ ϱ_1 ϱ_2. *is-inference-grounding* ι ι_G γ ϱ_1 ϱ_2 }

lemma *is-inference-grounding-inference-groundings*:
is-inference-grounding ι ι_G γ ϱ_1 ϱ_2 $\Longrightarrow$ $\iota_G \in$ *inference-groundings* ι
unfolding *inference-groundings-def*
by *blast*

lemma *inference*$_G$*-concl-in-clause-grounding*:
assumes $\iota_G \in$ *inference-groundings* ι
shows *concl-of* $\iota_G \in$ *clause-groundings typeof-fun* (*concl-of* ι)
proof−
obtain *premises*$_G$ *conlcusion*$_G$ **where**
ι_G: ι_G = *Infer premises*$_G$ *conlcusion*$_G$
using *Calculus.inference.exhaust* **by** *blast*

obtain *premises conclusion* $\mathcal{V}$ **where**
ι: ι = *Infer premises* (*conclusion*, $\mathcal{V}$)
using *Calculus.inference.exhaust*
by (*metis prod.collapse*)

obtain γ **where**
clause.is-ground (*conclusion* $\cdot$ γ)
conlcusion$_G$ = *clause.to-ground* (*conclusion* $\cdot$ γ)
welltyped$_c$ *typeof-fun* $\mathcal{V}$ *conclusion* $\wedge$ *welltyped*$_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}$ γ $\wedge$
term-subst.is-ground-subst γ $\wedge$ *all-types* $\mathcal{V}$
proof−

have $\bigwedge\gamma\ \varrho_1\ \varrho_2.$
$[\![\bigwedge\gamma.\ [\![$*clause.vars* (*conclusion* $\cdot\ \gamma$) = {}; *conlcusion*$_G$ = *clause.to-ground* (*conclusion* $\cdot\ \gamma$);
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *conclusion* $\wedge$
welltyped$_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}\ \gamma\ \wedge$
term-subst.is-ground-subst $\gamma\ \wedge$ *all-types* $\mathcal{V}]\!]$
$\Longrightarrow$ *thesis*;
Infer premises$_G$ *conlcusion*$_G$ $\in$ *ground.G-Inf*;
case premises of [] $\Rightarrow$ *False*
| [(*premise*, $\mathcal{V}'$)] $\Rightarrow$
term-subst.is-ground-subst $\gamma\ \wedge$
Infer premises$_G$ *conlcusion*$_G$ =
Infer [*clause.to-ground* (*premise* $\cdot\ \gamma$)] (*clause.to-ground* (*conclusion* $\cdot\ \gamma$)) $\wedge$
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise* $\wedge$
welltyped$_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}\ \gamma\ \wedge$
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *conclusion* $\wedge\ \mathcal{V} = \mathcal{V}'\ \wedge$ *all-types* $\mathcal{V}$
| [(*premise*, $\mathcal{V}'$), (*premise*$_1$, $\mathcal{V}_1$)] $\Rightarrow$
clause.is-renaming $\varrho_1\ \wedge$
clause.is-renaming $\varrho_2\ \wedge$
clause.vars (*premise*$_1$ $\cdot\ \varrho_1$) $\cap$ *clause.vars* (*premise* $\cdot\ \varrho_2$) = {} $\wedge$
term-subst.is-ground-subst $\gamma\ \wedge$
Infer premises$_G$ *conlcusion*$_G$ =
Infer [*clause.to-ground* (*premise* $\cdot\ \varrho_2\ \cdot\ \gamma$), *clause.to-ground* (*premise*$_1$ $\cdot\ \varrho_1\ \cdot\ \gamma$)]
(*clause.to-ground* (*conclusion* $\cdot\ \gamma$)) $\wedge$
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}_1$ *premise*$_1$ $\wedge$
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}'$ *premise* $\wedge$
welltyped$_\sigma$*-on* (*clause.vars conclusion*) *typeof-fun* $\mathcal{V}\ \gamma\ \wedge$
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *conclusion* $\wedge$
all-types $\mathcal{V}_1\ \wedge$ *all-types* $\mathcal{V}'\ \wedge$ *all-types* $\mathcal{V}$
| (*premise*, $\mathcal{V}'$) # (*premise*$_1$, $\mathcal{V}_1$) # *a* # *lista* $\Rightarrow$ *False*$]\!]$
$\Longrightarrow$ *thesis*
by(*auto simp*: *clause.is-ground-subst-is-ground split*: *list.splits*)
(*metis list-4-cases prod.exhaust-sel*)

then show *?thesis*
using *that assms*
unfolding *inference-groundings-def* ι ι_G *Calculus.inference.case*
by (*auto simp*: *is-inference-grounding-def*)
qed

then show *?thesis*
unfolding ι ι_G *clause-groundings-def*
by *auto*
qed

lemma *inference*$_G$*-red-in-clause-grounding-of-concl*:

assumes $\iota_G \in$ *inference-groundings* ι
shows $\iota_G \in$ *ground.Red-I* (*clause-groundings typeof-fun* (*concl-of* ι))
proof −
from *assms* **have** $\iota_G \in$ *ground.G-Inf*
unfolding *inference-groundings-def is-inference-grounding-def*
by *blast*

moreover have *concl-of* $\iota_G \in$ *clause-groundings typeof-fun* (*concl-of* ι)
using *assms inference*$_G$*-concl-in-clause-grounding*
by *auto*

ultimately show $\iota_G \in$ *ground.Red-I* (*clause-groundings typeof-fun* (*concl-of* ι))
using *ground.Red-I-of-Inf-to-N*
by *blast*
qed

lemma *obtain-welltyped-ground-subst*:
obtains γ :: ('*f*, '*v*) *subst* **and** $\mathcal{F}_G$ :: ('*f*, '*ty*) *fun-types*
where *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ γ *term-subst.is-ground-subst* γ
proof −

define γ :: ('*f*, '*v*) *subst* **where**
$\bigwedge x.\ \gamma\ x \equiv$ *Fun* (*SOME f. typeof-fun f* = ([], $\mathcal{V}$ *x*)) []

moreover have *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ γ
proof −
have $\bigwedge x.$ *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}$
(*Fun* (*SOME f. typeof-fun f* = ([], $\mathcal{V}$ *x*)) []) ($\mathcal{V}$ *x*)
by (*meson function-symbols list-all2-Nil someI-ex welltyped.Fun*)

then show *?thesis*
unfolding *welltyped*$_\sigma$*-def* γ*-def*
by *auto*
qed

moreover have *term-subst.is-ground-subst* γ
unfolding *term-subst.is-ground-subst-def* γ*-def*
by (*smt* (*verit*) *Nil-is-map-conv equals0D eval-term.simps*(*2*) *is-ground-iff is-ground-trm-iff-ident-forall-subs*

ultimately show *?thesis*
using *that*
by *blast*
qed

lemma *welltyped*$_\sigma$*-on-empty*: *welltyped*$_\sigma$*-on* {} $\mathcal{F}$ $\mathcal{V}$ σ
unfolding *welltyped*$_\sigma$*-on-def*
by *simp*

```
sublocale lifting:
  tiebreaker-lifting
    ⊥_F
    inferences
    ground.G-Bot
    ground.G-entails
    ground.G-Inf
    ground.GRed-I
    ground.GRed-F
    clause-groundings typeof-fun
    (Some ∘ inference-groundings)
    typed-tiebreakers
proof unfold-locales
 show ⊥_F ≠ {}
  using all-types'[OF variables]
  by blast
next
 fix bottom
 assume bottom ∈ ⊥_F

 then show clause-groundings typeof-fun bottom ≠ {}
  unfolding clause-groundings-def
  using welltyped_σ-Var
 proof −
  have ∃f. welltyped_σ-on (clause.vars {#}) typeof-fun (snd bottom) f ∧
     First-Order-Type-System.welltyped_c typeof-fun (snd bottom) {#} ∧
     term-subst.is-ground-subst f
    by (metis First-Order-Type-System.welltyped_c-def empty-clause-is-ground
ex-in-conv
     set-mset-eq-empty-iff term.obtain-ground-subst welltyped_σ-on-empty)

  then show {clause.to-ground (fst bottom · f) |f. term-subst.is-ground-subst f
    ∧ First-Order-Type-System.welltyped_c typeof-fun (snd bottom) (fst bottom)
    ∧ welltyped_σ-on (clause.vars (fst bottom)) typeof-fun (snd bottom) f
    ∧ all-types (snd bottom)} ≠ {}
   using ‹bottom ∈ ⊥_F› by force
 qed
next
 fix bottom
 assume bottom ∈ ⊥_F
 then show clause-groundings typeof-fun bottom ⊆ ground.G-Bot
  unfolding clause-groundings-def
  by clause-auto
next
 fix clause
 show clause-groundings typeof-fun clause ∩ ground.G-Bot ≠ {} ⟶ clause ∈ ⊥_F
  unfolding clause-groundings-def clause.to-ground-def clause.subst-def
 by (smt (verit) disjoint-insert(1) image-mset-is-empty-iff inf-bot-right mem-Collect-eq
```

prod.exhaust-sel)
next
fix ι :: ('*f*, '*v*, '*ty*) *typed-clause inference*

show *the* ((*Some* ∘ *inference-groundings*) ι) ⊆
ground.GRed-I (*clause-groundings typeof-fun* (*concl-of* ι))
using $inference_G$*-red-in-clause-grounding-of-concl*
by *auto*
next
show $\bigwedge clause_G$. *po-on* (*typed-tiebreakers* $clause_G$) *UNIV*
unfolding *po-on-def*
using *wellfounded-typed-tiebreakers*
by *simp*
next
show $\bigwedge clause_G$. *Restricted-Predicates.wfp-on* (*typed-tiebreakers* $clause_G$) *UNIV*
using *wellfounded-typed-tiebreakers*
by *simp*
qed

end

sublocale *first-order-superposition-calculus* ⊆
lifting-intersection
inferences
{{#}}
$select_{Gs}$
ground-superposition-calculus.G-Inf ($\prec_{tG}$)
λ-. *ground-superposition-calculus.G-entails*
ground-superposition-calculus.GRed-I ($\prec_{tG}$)
λ-. *ground-superposition-calculus.GRed-F*($\prec_{tG}$)
$\bot_F$
λ-. *clause-groundings typeof-fun*
$\lambda select_G$. *Some* ∘
(*grounded-first-order-superposition-calculus.inference-groundings* ($\prec_t$) $select_G$
typeof-fun)
typed-tiebreakers
proof(*unfold-locales*; (*intro ballI*)*?*)
show $select_{Gs} \neq \{\}$
using $select_G$*-simple*
unfolding $select_{Gs}$*-def*
by *blast*
next
fix $select_G$
assume $select_G \in select_{Gs}$

then interpret *grounded-first-order-superposition-calculus*
where $select_G = select_G$
apply *unfold-locales*

by(*simp add*: $select_{Gs}$*-def*)

show *consequence-relation ground.G-Bot ground.G-entails*
using *ground.consequence-relation-axioms*.

next
fix $select_G$
assume $select_G \in select_{Gs}$

then interpret *grounded-first-order-superposition-calculus*
where $select_G = select_G$
by *unfold-locales* (*simp add*: $select_{Gs}$*-def*)

show *tiebreaker-lifting*
$\bot_F$
inferences
ground.G-Bot
ground.G-entails
ground.G-Inf
ground.GRed-I
ground.GRed-F
(*clause-groundings typeof-fun*)
(*Some* $\circ$ *inference-groundings*)
typed-tiebreakers
by *unfold-locales*
qed

end
theory *First-Order-Superposition-Completeness*
imports
Ground-Superposition-Completeness
Grounded-First-Order-Superposition
HOL−ex.Sketch-and-Explore
begin

lemma $welltyped_\sigma$*-on-term*:
assumes $welltyped_\sigma$*-on* (*term.vars term*) $\mathcal{F}$ $\mathcal{V}$ γ
shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ *term* $\tau \longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*term* $\cdot t$ γ) τ
by (*simp add*: *assms* $welltyped_\sigma$*-on-welltyped*)

context *grounded-first-order-superposition-calculus*
begin

lemma *eq-resolution-lifting*:
fixes
$premise_G$ $conclusion_G$ :: *'f gatom clause* **and**
premise conclusion :: (*'f*, *'v*) *atom clause* **and**
γ :: (*'f*, *'v*) *subst*
defines

premise$_G$ [*simp*]: *premise*$_G$ ≡ *clause.to-ground* (*premise* · γ) **and**
conclusion$_G$ [*simp*]: *conclusion*$_G$ ≡ *clause.to-ground* (*conclusion* · γ)
assumes
premise-grounding: *clause.is-ground* (*premise* · γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* · γ) **and**
select: *clause.from-ground* (*select*$_G$ *premise*$_G$) = (*select premise*) · γ **and**
ground-eq-resolution: *ground.ground-eq-resolution premise*$_G$ *conclusion*$_G$ **and**
typing:
welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise*
term-subst.is-ground-subst γ
welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ
all-types $\mathcal{V}$
obtains *conclusion'*
where
eq-resolution (*premise*, $\mathcal{V}$) (*conclusion'*, $\mathcal{V}$)
Infer [*premise*$_G$] *conclusion*$_G$ ∈ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion'*, $\mathcal{V}$))
conclusion' · γ = *conclusion* · γ
using *ground-eq-resolution*
proof(*cases premise*$_G$ *conclusion*$_G$ *rule*: *ground.ground-eq-resolution.cases*)
case (*ground-eq-resolutionI literal*$_G$ *premise*$_G$*' term*$_G$)

have *premise-not-empty*: *premise* ≠ {#}
using
ground-eq-resolutionI(*1*)
empty-not-add-mset
clause-subst-empty
unfolding *premise*$_G$
by (*metis clause-from-ground-empty-mset clause.from-ground-inverse*)

have *premise* · γ = *clause.from-ground* (*add-mset literal*$_G$ (*clause.to-ground* (*conclusion* · γ)))
using
ground-eq-resolutionI(*1*)[*THEN arg-cong*, *of clause.from-ground*]
clause.to-ground-inverse[*OF premise-grounding*]
ground-eq-resolutionI(*4*)
unfolding *premise*$_G$ *conclusion*$_G$
by *metis*

also have ... = *add-mset* (*literal.from-ground literal*$_G$) (*conclusion* · γ)
unfolding *clause-from-ground-add-mset*
by (*simp add*: *conclusion-grounding*)

finally have *premise-γ*: *premise* · γ = *add-mset* (*literal.from-ground literal*$_G$) (*conclusion* · γ)**.**

let *?select*$_G$*-empty* = *select*$_G$ *premise*$_G$ = {#}
let *?select*$_G$*-not-empty* = *select*$_G$ *premise*$_G$ ≠ {#}

obtain *max-literal* **where** *max-literal*:
is-maximal$_l$ *max-literal premise*
is-maximal$_l$ (*max-literal* $\cdot l$ γ) (*premise* $\cdot$ γ)
using *is-maximal*$_l$*-ground-subst-stability*[*OF premise-not-empty premise-grounding*]
by *blast*

moreover then have *max-literal* $\in\#$ *premise*
using *maximal*$_l$*-in-clause* **by** *fastforce*

moreover have *max-literal-*γ: *max-literal* $\cdot l$ γ = *literal.from-ground* (*term*$_G$!$\approx$ *term*$_G$)
if *?select*$_G$*-empty*
proof−
have *ground.is-maximal-lit literal*$_G$ *premise*$_G$
using *ground-eq-resolutionI*(*3*) *that maximal-lit-in-clause*
by (*metis empty-iff set-mset-empty*)

then show *?thesis*
using *max-literal*(*2*) *unique-maximal-in-ground-clause*[*OF premise-grounding*]

unfolding
ground-eq-resolutionI(*2*)
is-maximal-lit-iff-is-maximal$_l$
premise$_G$
clause.to-ground-inverse[*OF premise-grounding*]
by *blast*
qed

moreover obtain *selected-literal* **where**
selected-literal $\cdot l$ γ = *literal.from-ground* (*term*$_G$!$\approx$ *term*$_G$) **and**
is-maximal$_l$ *selected-literal* (*select premise*)
if *?select*$_G$*-not-empty*
proof−
have *ground.is-maximal-lit literal*$_G$ (*select*$_G$ *premise*$_G$) **if** *?select*$_G$*-not-empty*
using *ground-eq-resolutionI*(*3*) *that*
by *blast*

then show *?thesis*
using
that
select
unique-maximal-in-ground-clause[*OF select-subst*(*1*)[*OF premise-grounding*]]
is-maximal$_l$*-ground-subst-stability*[*OF - select-subst*(*1*)[*OF premise-grounding*]]
unfolding
ground-eq-resolutionI(*2*)
premise$_G$
is-maximal-lit-iff-is-maximal$_l$
by (*metis* (*full-types*) *clause-subst-empty*(*2*) *clause.from-ground-inverse clause-to-ground-empty-mset*)
qed

moreover then have *selected-literal* $\in\#$ *premise* **if** *?select$_G$-not-empty*
by (*meson that maximal$_l$-in-clause mset-subset-eqD select-subset*)

ultimately obtain *literal* **where**
literal-γ: *literal* $\cdot l\ \gamma$ = *literal.from-ground* (*term$_G$* !$\approx$ *term$_G$*) **and**
literal-in-premise: *literal* $\in\#$ *premise* **and**
literal-selected: *?select$_G$-not-empty* $\Longrightarrow$ *is-maximal$_l$ literal* (*select premise*) **and**
literal-max: *?select$_G$-empty* $\Longrightarrow$ *is-maximal$_l$ literal premise*
by *blast*

have *literal-grounding*: *literal.is-ground* (*literal* $\cdot l\ \gamma$)
using *literal-γ*
by *simp*

from *literal-γ* **obtain** *term term'* **where**
literal: *literal* = *term* !$\approx$ *term'*
using *subst-polarity-stable literal-from-ground-polarity-stable*
by (*metis literal.collapse*(*2*) *literal.disc*(*2*) *uprod-exhaust*)

have *literal$_G$*:
literal.from-ground literal$_G$ = (*term* !$\approx$ *term'*) $\cdot l\ \gamma$
literal$_G$ = *literal.to-ground* ((*term* !$\approx$ *term'*) $\cdot l\ \gamma$)
using *literal-γ literal ground-eq-resolutionI*(*2*)
by *simp-all*

obtain *conclusion'* **where** *conclusion'*: *premise* = *add-mset literal conclusion'*
using *multi-member-split*[*OF literal-in-premise*]
by *blast*

have *term* $\cdot t\ \gamma$ = *term'* $\cdot t\ \gamma$
using *literal-γ*
unfolding *literal subst-literal*(*2*) *atom.subst-def literal.from-ground-def atom.from-ground-def*
by *simp*

moreover obtain τ **where** *welltyped typeof-fun* $\mathcal{V}$ *term* τ *welltyped typeof-fun* $\mathcal{V}$ *term'* τ
using *typing*(*1*)
unfolding *conclusion' literal welltyped$_c$-def welltyped$_l$-def welltyped$_a$-def*
by *auto*

ultimately obtain μ σ **where** μ:
term-subst.is-imgu μ {{*term*, *term'*}}
$\gamma = \mu \odot \sigma$
welltyped-imgu' typeof-fun $\mathcal{V}$ *term term'* μ
using *welltyped-imgu'-exists*
by *meson*

have *conclusion′-γ*: *conclusion′* · *γ* = *conclusion* · *γ*
  using *premise-γ*
 unfolding *conclusion′ ground-eq-resolutionI*(*2*) *literal-γ*[*symmetric*] *subst-clause-add-mset*
  by *simp*

have *eq-resolution*: *eq-resolution* (*premise*, 𝒱) (*conclusion′* · *μ*, 𝒱)
proof (*rule eq-resolutionI*)
  show *premise* = *add-mset literal conclusion′*
    using *conclusion′*.
next
  show *literal* = *term* !≈ *term′*
    using *literal*.
next
  show *term-subst.is-imgu* *μ* {{*term*, *term′*}}
    using *μ*(*1*).
next
  show *select premise* = {#} ∧ *is-maximal*$_l$ (*literal* ·*l* *μ*) (*premise* · *μ*)
    ∨ *is-maximal*$_l$ (*literal* ·*l* *μ*) ((*select premise*) · *μ*)
  proof(*cases ?select*$_G$*-empty*)
    case *select*$_G$*-empty*: *True*

    then have *max-literal* ·*l* *γ* = *literal* ·*l* *γ*
      by (*simp add*: *max-literal-γ literal-γ*)

    then have *literal-γ-is-maximal*: *is-maximal*$_l$ (*literal* ·*l* *γ*) (*premise* · *γ*)
      using *max-literal*(*2*) **by** *simp*

    have *literal-μ-in-premise*: *literal* ·*l* *μ* ∈# *premise* · *μ*
      by (*simp add*: *clause.subst-in-to-set-subst literal-in-premise*)

    have *is-maximal*$_l$ (*literal* ·*l* *μ*) (*premise* · *μ*)
      using *is-maximal*$_l$*-ground-subst-stability′*[*OF*
        literal-μ-in-premise
        premise-grounding[*unfolded* *μ*(*2*) *clause.subst-comp-subst*]
     literal-γ-is-maximal[*unfolded* *μ*(*2*) *clause.subst-comp-subst literal.subst-comp-subst*]
        ].

    then show *?thesis*
      using *select select*$_G$*-empty*
      by *clause-auto*
  next
    case *False*

    have *selected-grounding*: *clause.is-ground* (*select premise* · *μ* · *σ*)
      using *select-subst*(*1*)[*OF premise-grounding*]
      unfolding *μ*(*2*) *clause.subst-comp-subst*.

    note *selected-subst* =
    literal-selected[*OF False*, *THEN maximal*$_l$*-in-clause*, *THEN clause.subst-in-to-set-subst*]

have *is-maximal*$_l$ (*literal* ·*l* γ) (*select premise* · γ)
using *False ground-eq-resolutionI*(*3*)
unfolding *ground-eq-resolutionI*(*2*) *is-maximal-lit-iff-is-maximal*$_l$ *literal-*γ *select*
by *presburger*

then have *is-maximal*$_l$ (*literal* ·*l* μ) (*select premise* · μ)
unfolding μ(*2*) *clause.subst-comp-subst literal.subst-comp-subst*
using *is-maximal*$_l$*-ground-subst-stability'*[*OF selected-subst selected-grounding*]
by *argo*

with *False* **show** *?thesis*
by *blast*
qed
next
show *welltyped-imgu' typeof-fun* $\mathcal{V}$ *term term'* μ
using μ(*3*)**.**
next
show *conclusion'* · μ = *conclusion'* · μ **..**
qed

have *term-subst.is-idem* μ
using μ(*1*)
by (*simp add*: *term-subst.is-imgu-iff-is-idem-and-is-mgu*)

then have μ*-*γ: $\mu \odot \gamma = \gamma$
unfolding μ(*2*) *term-subst.is-idem-def*
by (*metis subst-compose-assoc*)

have *vars-conclusion'*: *clause.vars* (*conclusion'* · μ) $\subseteq$ *clause.vars premise*
using *vars-clause-imgu*[*OF* μ(*1*)]
unfolding *conclusion' literal*
by *clause-auto*

have *conclusion'* · μ · γ = *conclusion* · γ
using *conclusion'-*γ
unfolding *clause.subst-comp-subst*[*symmetric*] μ*-*γ**.**

moreover have
Infer [*premise*$_G$] *conclusion*$_G$ $\in$ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion'* · μ, $\mathcal{V}$))
unfolding *inference-groundings-def mem-Collect-eq*
proof −
have *Infer* [*premise*$_G$] *conclusion*$_G$ $\in$ *ground.G-Inf*
unfolding *ground.G-Inf-def*
using *ground-eq-resolution* **by** *blast*

then have $\exists\, \varrho_1\ \varrho_2$. *is-inference-grounding*

(*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion*$'$ · μ, $\mathcal{V}$))
(*Infer* [*premise*$_G$] *conclusion*$_G$) γ ϱ_1 ϱ_2
unfolding *is-inference-grounding-def Calculus.inference.case list.case prod.case*
using *typing*
by (*smt* (*verit*) *calculation conclusion*$_G$ *eq-resolution eq-resolution-preserves-typing premise*$_G$
vars-conclusion$'$ *welltyped*$_\sigma$*-on-subset*)

thus $\exists\, \iota_G\ \gamma\ \varrho_1\ \varrho_2$. *Infer* [*premise*$_G$] *conclusion*$_G$ = ι_G $\wedge$
is-inference-grounding (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion*$'$ · μ, $\mathcal{V}$)) ι_G γ ϱ_1 ϱ_2
by *iprover*
qed

ultimately show *?thesis*
using *that*[*OF eq-resolution*]
by *blast*
qed

lemma *eq-factoring-lifting*:
fixes
premise$_G$ *conclusion*$_G$ :: $'f$ *gatom clause* **and**
premise conclusion :: ($'f$, $'v$) *atom clause* **and**
γ :: ($'f$, $'v$) *subst*
defines
premise$_G$ [*simp*]: *premise*$_G$ $\equiv$ *clause.to-ground* (*premise* · γ) **and**
conclusion$_G$ [*simp*]: *conclusion*$_G$ $\equiv$ *clause.to-ground* (*conclusion* · γ)
assumes
premise-grounding: *clause.is-ground* (*premise* · γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* · γ) **and**
select: *clause.from-ground* (*select*$_G$ *premise*$_G$) = (*select premise*) · γ **and**
ground-eq-factoring: *ground.ground-eq-factoring premise*$_G$ *conclusion*$_G$ **and**
typing:
welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise*
term-subst.is-ground-subst γ
welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ
all-types $\mathcal{V}$
obtains *conclusion*$'$
where
eq-factoring (*premise*, $\mathcal{V}$) (*conclusion*$'$, $\mathcal{V}$)
Infer [*premise*$_G$] *conclusion*$_G$ $\in$ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion*$'$, $\mathcal{V}$))
conclusion$'$ · γ = *conclusion* · γ
using *ground-eq-factoring*
proof(*cases premise*$_G$ *conclusion*$_G$ *rule*: *ground.ground-eq-factoring.cases*)
case (*ground-eq-factoringI literal*$_{G1}$ *literal*$_{G2}$ *premise*$'_G$ *term*$_{G1}$ *term*$_{G2}$ *term*$_{G3}$)

have *premise-not-empty*: *premise* $\neq$ {#}
using *ground-eq-factoringI*(*1*) *empty-not-add-mset clause-subst-empty premise*$_G$
by (*metis clause-from-ground-empty-mset clause.from-ground-inverse*)

have *select-empty*: *select premise* = {#}
using *ground-eq-factoringI*(*4*) *select clause-subst-empty*
by *clause-auto*

have *premise*-γ: *premise* · γ = *clause.from-ground* (*add-mset literal*$_{G1}$ (*add-mset literal*$_{G2}$ *premise*$'_G$))
using *ground-eq-factoringI*(*1*) *premise*$_G$
by (*metis premise-grounding clause.to-ground-inverse*)

obtain *literal*$_1$ **where** *literal*$_1$*-maximal*:
is-maximal$_l$ *literal*$_1$ *premise*
is-maximal$_l$ (*literal*$_1$ ·*l* γ) (*premise* · γ)
using *is-maximal*$_l$*-ground-subst-stability*[*OF premise-not-empty premise-grounding*]
by *blast*

have *max-ground-literal*: *is-maximal*$_l$ (*literal.from-ground* (*term*$_{G1}$ ≈ *term*$_{G2}$)) (*premise* · γ)
using *ground-eq-factoringI*(*5*)
unfolding
is-maximal-lit-iff-is-maximal$_l$
ground-eq-factoringI(*2*)
premise$_G$
clause.to-ground-inverse[*OF premise-grounding*].

have *literal*$_1$-γ: *literal*$_1$ ·*l* γ = *literal.from-ground literal*$_{G1}$
using
unique-maximal-in-ground-clause[*OF premise-grounding literal*$_1$*-maximal*(*2*) *max-ground-literal*]
ground-eq-factoringI(*2*)
by *blast*

then have *is-pos literal*$_1$
unfolding *ground-eq-factoringI*(*2*)
using *literal-from-ground-stable subst-pos-stable*
by (*metis literal.disc*(*1*))

with *literal*$_1$-γ **obtain** *term*$_1$ *term*$_1{}'$ **where**
literal$_1$*-terms*: *literal*$_1$ = *term*$_1$ ≈ *term*$_1{}'$ **and**
term$_{G1}$*-term*$_1$: *term.from-ground term*$_{G1}$ = *term*$_1$ ·*t* γ
unfolding *ground-eq-factoringI*(*2*)
by *clause-simp*

obtain *premise*$''$ **where** *premise*$''$: *premise* = *add-mset literal*$_1$ *premise*$''$
using *maximal*$_l$*-in-clause*[*OF literal*$_1$*-maximal*(*1*)]
by (*meson multi-member-split*)

then have *premise*$''$-γ: *premise*$''$ · γ = *add-mset* (*literal.from-ground literal*$_{G2}$) (*clause.from-ground premise*$'_G$)

using *premise-*γ
unfolding *clause-from-ground-add-mset* $literal_1$-γ[*symmetric*]
by (*simp add*: *subst-clause-add-mset*)

then obtain $literal_2$ **where** $literal_2$:
$literal_2 \cdot l\ \gamma = literal.from\text{-}ground\ literal_{G2}$
$literal_2 \in\#\ premise''$
unfolding *clause.subst-def*
using *msed-map-invR* **by** *force*

then have *is-pos* $literal_2$
unfolding *ground-eq-factoringI*(*3*)
using *literal-from-ground-stable subst-pos-stable*
by (*metis literal.disc*(*1*))

with $literal_2$ **obtain** $term_2$ $term_2'$ **where**
$literal_2$-*terms*: $literal_2 = term_2 \approx term_2'$ **and**
$term_{G1}$-$term_2$: $term.from\text{-}ground\ term_{G1} = term_2 \cdot t\ \gamma$
unfolding *ground-eq-factoringI*(*3*)
by *clause-simp*

have $term_{G2}$-$term_1'$: $term.from\text{-}ground\ term_{G2} = term_1' \cdot t\ \gamma$
using $literal_1$-γ $term_{G1}$-$term_1$
unfolding
$literal_1$-*terms*
ground-eq-factoringI(*2*)
apply *clause-simp*
by *auto*

have $term_{G3}$-$term_2'$: $term.from\text{-}ground\ term_{G3} = term_2' \cdot t\ \gamma$
using $literal_2$ $term_{G1}$-$term_2$
unfolding
$literal_2$-*terms*
ground-eq-factoringI(*3*)
by *clause-auto*

obtain *premise*$'$ **where** *premise*: *premise* = *add-mset* $literal_1$ (*add-mset* $literal_2$ *premise*$'$)
using $literal_2$(*2*) $maximal_l$-*in-clause*[*OF* $literal_1$-*maximal*(*1*)] *premise*$''$
by (*metis multi-member-split*)

then have *premise*$'$-γ: $premise' \cdot \gamma = clause.from\text{-}ground\ premise'_G$
using *premise*$''$-γ *premise*$''$
unfolding $literal_2$(*1*)[*symmetric*]
by (*simp add*: *subst-clause-add-mset*)

have $term_1$-$term_2$: $term_1 \cdot t\ \gamma = term_2 \cdot t\ \gamma$
using $term_{G1}$-$term_1$ $term_{G1}$-$term_2$
by *argo*

moreover obtain τ **where** *welltyped typeof-fun* $\mathcal{V}$ *term*$_1$ τ *welltyped typeof-fun* $\mathcal{V}$ *term*$_2$ τ
proof−
have *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ (*premise* $\cdot$ γ)
using *typing*
using *welltyped*$_\sigma$*-on-welltyped*$_c$ **by** *blast*

then obtain τ **where** *welltyped typeof-fun* $\mathcal{V}$ (*term.from-ground term*$_{G1}$) τ
unfolding *premise-*γ *ground-eq-factoringI*
by *clause-simp*

then have *welltyped typeof-fun* $\mathcal{V}$ (*term*$_1$ $\cdot t$ γ) τ *welltyped typeof-fun* $\mathcal{V}$ (*term*$_2$ $\cdot t$ γ) τ
using *term*$_{G1}$*-term*$_1$ *term*$_{G1}$*-term*$_2$
by *metis+*

then have *welltyped typeof-fun* $\mathcal{V}$ *term*$_1$ τ *welltyped typeof-fun* $\mathcal{V}$ *term*$_2$ τ
using *typing*(*3*) *welltyped*$_\sigma$*-on-term*
unfolding *welltyped*$_\sigma$*-on-def premise literal*$_1$*-terms literal*$_2$*-terms*
apply *clause-simp*
by (*metis UnCI welltyped*$_\sigma$*-on-def welltyped*$_\sigma$*-on-term*)+

then show *?thesis*
using *that*
by *blast*
qed

ultimately obtain μ σ **where** μ:
term-subst.is-imgu μ $\{\{$*term*$_1$, *term*$_2\}\}$
$\gamma = \mu \odot \sigma$
welltyped-imgu$'$ *typeof-fun* $\mathcal{V}$ *term*$_1$ *term*$_2$ μ
using *welltyped-imgu*$'$*-exists*
by *meson*

let *?conclusion*$'$ = *add-mset* (*term*$_1$ $\approx$ *term*$_2{}'$) (*add-mset* (*term*$_1{}'$!$\approx$ *term*$_2{}'$) *premise*$'$)

have *eq-factoring*: *eq-factoring* (*premise*, $\mathcal{V}$) (*?conclusion*$'$ $\cdot$ μ, $\mathcal{V}$)
proof (*rule eq-factoringI*)
show *premise* = *add-mset literal*$_1$ (*add-mset literal*$_2$ *premise*$'$)
using *premise*.
next
show *literal*$_1$ = *term*$_1$ $\approx$ *term*$_1{}'$
using *literal*$_1$*-terms*.
next
show *literal*$_2$ = *term*$_2$ $\approx$ *term*$_2{}'$
using *literal*$_2$*-terms*.
next

show *select premise* = {#}
using *select-empty*.
next
have *literal*$_1$*-μ-in-premise*: *literal*$_1$ $\cdot l$ μ ∈# *premise* $\cdot$ μ
using *literal*$_1$*-maximal(1)* *clause.subst-in-to-set-subst* *maximal*$_l$*-in-clause* **by** *blast*

have *is-maximal*$_l$ (*literal*$_1$ $\cdot l$ μ) (*premise* $\cdot$ μ)
using *is-maximal*$_l$*-ground-subst-stability'*[*OF*
literal$_1$*-μ-in-premise*
premise-grounding[*unfolded* μ(*2*) *clause.subst-comp-subst*]
literal$_1$*-maximal*(*2*)[*unfolded* μ(*2*) *clause.subst-comp-subst* *literal.subst-comp-subst*]
].

then show *is-maximal*$_l$ (*literal*$_1$ $\cdot l$ μ) (*premise* $\cdot$ μ)
by *blast*
next
have *term-groundings*: *term.is-ground* (*term*$_1$' $\cdot t$ μ $\cdot t$ σ) *term.is-ground* (*term*$_1$ $\cdot t$ μ $\cdot t$ σ)
unfolding
term-subst.subst-comp-subst[*symmetric*]
μ(*2*)[*symmetric*]
term$_{G1}$*-term*$_1$[*symmetric*]
term$_{G2}$*-term*$_1$'[*symmetric*]
using *term.ground-is-ground*
by *simp-all*

have *term*$_1$' $\cdot t$ μ $\cdot t$ σ $\prec_t$ *term*$_1$ $\cdot t$ μ $\cdot t$ σ
using *ground-eq-factoringI*(*6*)[*unfolded*
less$_{tG}$*-def*
term$_{G1}$*-term*$_1$
term$_{G2}$*-term*$_1$'
μ(*2*)
term-subst.subst-comp-subst
].

then show ¬ *term*$_1$ $\cdot t$ μ $\preceq_t$ *term*$_1$' $\cdot t$ μ
using *less*$_t$*-less-eq*$_t$*-ground-subst-stability*[*OF term-groundings*]
by *blast*
next
show *term-subst.is-imgu* μ {{*term*$_1$, *term*$_2$}}
using μ(*1*).
next
show *welltyped-imgu'* *typeof-fun* $\mathcal{V}$ *term*$_1$ *term*$_2$ μ
using μ(*3*).
next
show *?conclusion'* $\cdot$ μ = *?conclusion'* $\cdot$ μ
..
qed

have *term-subst.is-idem* μ
  using $\mu(1)$
  by (*simp add*: *term-subst.is-imgu-iff-is-idem-and-is-mgu*)

then have μ-γ: $\mu \odot \gamma = \gamma$
  unfolding $\mu(2)$ *term-subst.is-idem-def*
  by (*metis subst-compose-assoc*)

have *vars-conclusion'*: *clause.vars* (*?conclusion'* $\cdot$ μ) $\subseteq$ *clause.vars premise*
  using *vars-clause-imgu*[*OF* $\mu(1)$] *vars-term-imgu*[*OF* $\mu(1)$]
  unfolding *premise* $literal_1$*-terms* $literal_2$*-terms*
  by *clause-auto*

have *conclusion* $\cdot$ γ =
  add-mset (*term.from-ground* $term_{G2}$!≈ *term.from-ground* $term_{G3}$)
  (*add-mset* (*term.from-ground* $term_{G1}$ ≈ *term.from-ground* $term_{G3}$) (*clause.from-ground* $premise'_G$))
  using *ground-eq-factoringI*(*7*) *clause.to-ground-inverse*[*OF conclusion-grounding*]
  unfolding *atom-from-ground-term-from-ground*[*symmetric*]
  literal-from-ground-atom-from-ground[*symmetric*] *clause-from-ground-add-mset*[*symmetric*]
  by *simp*

then have *conclusion*-γ:
  conclusion $\cdot$ γ = *add-mset* ($term_1$ ≈ $term_2'$) (*add-mset* ($term_1'$!≈ $term_2'$) *premise'*) $\cdot$ γ
  unfolding
    $term_{G2}$-$term_1'$
    $term_{G3}$-$term_2'$
    $term_{G1}$-$term_1$
    premise'-γ[*symmetric*]
  by(*clause-simp simp*: *add-mset-commute*)

then have *?conclusion'* $\cdot$ μ $\cdot$ γ = *conclusion* $\cdot$ γ
  by (*metis* μ-γ *clause.subst-comp-subst*)

moreover have
  Infer [$premise_G$] $conclusion_G$ $\in$ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*?conclusion'* $\cdot$ μ, $\mathcal{V}$))
  unfolding *inference-groundings-def mem-Collect-eq*
proof −
  have *Infer* [$premise_G$] $conclusion_G$ $\in$ *ground.G-Inf*
    unfolding *ground.G-Inf-def*
    using *ground-eq-factoring conclusion-grounding premise-grounding*
    by *blast*

  then have $\exists\, \varrho_1\ \varrho_2$. *is-inference-grounding*
    (*Infer* [(*premise*, $\mathcal{V}$)] (*?conclusion'* $\cdot$ μ, $\mathcal{V}$))
    (*Infer* [$premise_G$] $conclusion_G$) γ ϱ_1 ϱ_2

unfolding *is-inference-grounding-def Calculus.inference.case list.case prod.case*
using *typing*
by (*smt* (*verit*) *calculation conclusion*$_G$ *eq-factoring eq-factoring-preserves-typing premise*$_G$
vars-conclusion′ welltyped$_\sigma$*-on-subset*)

thus $\exists\,\iota_G\ \gamma\ \varrho_1\ \varrho_2$. *Infer* [*premise*$_G$] *conclusion*$_G$ = ι_G ∧
is-inference-grounding (*Infer* [(*premise*, $\mathcal{V}$)] (*?conclusion′* · μ, $\mathcal{V}$)) $\iota_G\ \gamma\ \varrho_1\ \varrho_2$
by *iprover*
qed

ultimately show *?thesis*
using *that*[*OF eq-factoring*]
by *blast*
qed

lemma *if-subst-sth* [*clause-simp*]: (*if b then Pos else Neg*) *atom* ·*l* ϱ =
(*if b then Pos else Neg*) (*atom* ·*a* ϱ)
by *clause-auto*

lemma *superposition-lifting*:
fixes
premise$_{G1}$ *premise*$_{G2}$ *conclusion*$_G$ :: ′*f gatom clause* **and**
premise$_1$ *premise*$_2$ *conclusion* :: (′*f*, ′*v*) *atom clause* **and**
$\gamma\ \varrho_1\ \varrho_2$ :: (′*f*, ′*v*) *subst* **and**
$\mathcal{V}_1\ \mathcal{V}_2$
defines
premise$_{G1}$ [*simp*]: *premise*$_{G1}$ ≡ *clause.to-ground* (*premise*$_1$ · ϱ_1 · γ) **and**
premise$_{G2}$ [*simp*]: *premise*$_{G2}$ ≡ *clause.to-ground* (*premise*$_2$ · ϱ_2 · γ) **and**
conclusion$_G$ [*simp*]: *conclusion*$_G$ ≡ *clause.to-ground* (*conclusion* · γ) **and**
premise-groundings [*simp*]:
premise-groundings ≡ *clause-groundings typeof-fun* (*premise*$_1$, $\mathcal{V}_1$) ∪
clause-groundings typeof-fun (*premise*$_2$, $\mathcal{V}_2$) **and**
ι_G [*simp*]: ι_G ≡ *Infer* [*premise*$_{G2}$, *premise*$_{G1}$] *conclusion*$_G$
assumes
renaming:
term-subst.is-renaming ϱ_1
term-subst.is-renaming ϱ_2
clause.vars (*premise*$_1$ · ϱ_1) ∩ *clause.vars* (*premise*$_2$ · ϱ_2) = {} **and**
premise$_1$*-grounding*: *clause.is-ground* (*premise*$_1$ · ϱ_1 · γ) **and**
premise$_2$*-grounding*: *clause.is-ground* (*premise*$_2$ · ϱ_2 · γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* · γ) **and**
select:
clause.from-ground (*select*$_G$ *premise*$_{G1}$) = (*select premise*$_1$) · ϱ_1 · γ
clause.from-ground (*select*$_G$ *premise*$_{G2}$) = (*select premise*$_2$) · ϱ_2 · γ **and**
ground-superposition: *ground.ground-superposition premise*$_{G2}$ *premise*$_{G1}$ *conclusion*$_G$ **and**

non-redundant: $\iota_G \notin$ *ground.Red-I premise-groundings* **and**
typing:
welltyped$_c$ typeof-fun $\mathcal{V}_1$ *premise*$_1$
welltyped$_c$ typeof-fun $\mathcal{V}_2$ *premise*$_2$
term-subst.is-ground-subst γ
welltyped$_\sigma$-on (*clause.vars premise*$_1$) *typeof-fun* $\mathcal{V}_1$ ($\varrho_1 \odot \gamma$)
welltyped$_\sigma$-on (*clause.vars premise*$_2$) *typeof-fun* $\mathcal{V}_2$ ($\varrho_2 \odot \gamma$)
welltyped$_\sigma$-on (*clause.vars premise*$_1$) *typeof-fun* $\mathcal{V}_1$ ϱ_1
welltyped$_\sigma$-on (*clause.vars premise*$_2$) *typeof-fun* $\mathcal{V}_2$ ϱ_2
all-types $\mathcal{V}_1$ *all-types* $\mathcal{V}_2$
obtains *conclusion′* $\mathcal{V}_3$
where
superposition (*premise*$_2$, $\mathcal{V}_2$) (*premise*$_1$, $\mathcal{V}_1$) (*conclusion′*, $\mathcal{V}_3$)
$\iota_G \in$ *inference-groundings* (*Infer* [(*premise*$_2$, $\mathcal{V}_2$), (*premise*$_1$, $\mathcal{V}_1$)] (*conclusion′*, $\mathcal{V}_3$))
conclusion′ $\cdot\ \gamma =$ *conclusion* $\cdot\ \gamma$
using *ground-superposition*
proof(*cases premise$_{G2}$ premise$_{G1}$ conclusion$_G$ rule*: *ground.ground-superposition.cases*)
case (*ground-superpositionI*
literal$_{G1}$
premise$_{G1}$′
literal$_{G2}$
premise$_{G2}$′
$\mathcal{P}_G$
context$_G$
term$_{G1}$
term$_{G2}$
term$_{G3}$
)

have *premise*$_1$*-not-empty*: *premise*$_1 \neq \{\#\}$
using *ground-superpositionI*(*1*) *empty-not-add-mset clause-subst-empty premise$_{G1}$*
by (*metis clause-from-ground-empty-mset clause.from-ground-inverse*)

have *premise*$_2$*-not-empty*: *premise*$_2 \neq \{\#\}$
using *ground-superpositionI*(*2*) *empty-not-add-mset clause-subst-empty premise$_{G2}$*
by (*metis clause-from-ground-empty-mset clause.from-ground-inverse*)

have *premise*$_1$-γ: *premise*$_1 \cdot \varrho_1 \cdot \gamma =$ *clause.from-ground* (*add-mset literal$_{G1}$ premise$_{G1}$′*)
using *ground-superpositionI*(*1*) *premise$_{G1}$*
by (*metis premise*$_1$*-grounding clause.to-ground-inverse*)

have *premise*$_2$-γ: *premise*$_2 \cdot \varrho_2 \cdot \gamma =$ *clause.from-ground* (*add-mset literal$_{G2}$ premise$_{G2}$′*)
using *ground-superpositionI*(*2*) *premise$_{G2}$*
by (*metis premise*$_2$*-grounding clause.to-ground-inverse*)

let *?select$_G$-empty* = *select$_G$* (*clause.to-ground* (*premise*$_1 \cdot \varrho_1 \cdot \gamma$)) $= \{\#\}$

let *?select$_G$-not-empty* = *select$_G$* (*clause.to-ground* (*premise$_1$* · ϱ_1 · γ)) ≠ {#}

have *pos-literal$_{G1}$-is-strictly-maximal$_l$*:
is-strictly-maximal$_l$ (*literal.from-ground literal$_{G1}$*) (*premise$_1$* · ϱ_1 ⊙ γ) **if** $\mathcal{P}_G$ = *Pos*
using *ground-superpositionI*(*9*) *that*
unfolding *is-strictly-maximal$_{Gl}$-iff-is-strictly-maximal$_l$*
by(*simp add*: *premise$_1$-grounding*)

have *neg-literal$_{G1}$-is-maximal$_l$*:
is-maximal$_l$ (*literal.from-ground literal$_{G1}$*) (*premise$_1$* · ϱ_1 ⊙ γ) **if** *?select$_G$-empty*
using
that
ground-superpositionI(*9*)
is-maximal$_l$-if-is-strictly-maximal$_l$
is-maximal$_l$-empty
premise$_1$-γ
unfolding
is-maximal-lit-iff-is-maximal$_l$
is-strictly-maximal$_{Gl}$-iff-is-strictly-maximal$_l$
ground-superpositionI(*1*)
apply *clause-auto*
by (*metis premise$_1$-γ clause-from-ground-empty-mset clause.from-ground-inverse*)

obtain *pos-literal$_1$* **where**
is-strictly-maximal$_l$ pos-literal$_1$ premise$_1$
pos-literal$_1$ ·*l* ϱ_1 ⊙ γ = *literal.from-ground literal$_{G1}$*
if $\mathcal{P}_G$ = *Pos*
using *is-strictly-maximal$_l$-ground-subst-stability*[*OF*
premise$_1$-grounding[*folded clause.subst-comp-subst*]
pos-literal$_{G1}$-is-strictly-maximal$_l$
]
by *blast*

moreover then have *pos-literal$_1$* ∈# *premise$_1$* **if** $\mathcal{P}_G$ = *Pos*
using *that strictly-maximal$_l$-in-clause* **by** *fastforce*

moreover obtain *neg-max-literal$_1$* **where**
is-maximal$_l$ neg-max-literal$_1$ premise$_1$
neg-max-literal$_1$ ·*l* ϱ_1 ⊙ γ = *literal.from-ground literal$_{G1}$*
if $\mathcal{P}_G$ = *Neg ?select$_G$-empty*
using
is-maximal$_l$-ground-subst-stability[*OF*
premise$_1$-not-empty
premise$_1$-grounding[*folded clause.subst-comp-subst*]
]
neg-literal$_{G1}$-is-maximal$_l$
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*9*) *clause.comp-subst.left.monoid-action-compatibility unique-maximal-in-ground-clause*)

moreover then have *neg-max-literal*$_1$ $\in\#$ *premise*$_1$ **if** $\mathcal{P}_G$ = *Neg ?select*$_G$*-empty*
using *that maximal*$_l$*-in-clause* **by** *fastforce*

moreover obtain *neg-selected-literal*$_1$ **where**
is-maximal$_l$ *neg-selected-literal*$_1$ (*select premise*$_1$)
neg-selected-literal$_1$ $\cdot l$ $\varrho_1 \odot \gamma$ = *literal.from-ground literal*$_{G1}$
if $\mathcal{P}_G$ = *Neg ?select*$_G$*-not-empty*
proof−
have *ground.is-maximal-lit literal*$_{G1}$ (*select*$_G$ *premise*$_{G1}$) **if** $\mathcal{P}_G$ = *Neg ?select*$_G$*-not-empty*
using *ground-superpositionI*(*9*) *that*
by *simp*

then show *?thesis*
using
that
select(*1*)
unique-maximal-in-ground-clause
is-maximal$_l$*-ground-subst-stability*
unfolding *premise*$_{G1}$ *is-maximal-lit-iff-is-maximal*$_l$
by (*metis* (*mono-tags*, *lifting*) *clause.comp-subst.monoid-action-compatibility*
clause-subst-empty(*2*) *clause.ground-is-ground image-mset-is-empty-iff*
clause.from-ground-def)
qed

moreover then have *neg-selected-literal*$_1$ $\in\#$ *premise*$_1$ **if** $\mathcal{P}_G$ = *Neg ?select*$_G$*-not-empty*

using *that*
by (*meson maximal*$_l$*-in-clause mset-subset-eqD select-subset*)

ultimately obtain *literal*$_1$ **where**
literal$_1$*-*γ: *literal*$_1$ $\cdot l$ $\varrho_1 \odot \gamma$ = *literal.from-ground literal*$_{G1}$ **and**
literal$_1$*-in-premise*$_1$: *literal*$_1$ $\in\#$ *premise*$_1$ **and**
literal$_1$*-is-strictly-maximal*: $\mathcal{P}_G$ = *Pos* $\Longrightarrow$ *is-strictly-maximal*$_l$ *literal*$_1$ *premise*$_1$
and
literal$_1$*-is-maximal*: $\mathcal{P}_G$ = *Neg* $\Longrightarrow$ *?select*$_G$*-empty* $\Longrightarrow$ *is-maximal*$_l$ *literal*$_1$
premise$_1$ **and**
literal$_1$*-selected*: $\mathcal{P}_G$ = *Neg* $\Longrightarrow$ *?select*$_G$*-not-empty* $\Longrightarrow$ *is-maximal*$_l$ *literal*$_1$
(*select premise*$_1$)
by (*metis ground-superpositionI*(*9*) *literals-distinct*(*1*))

then have *literal*$_1$*-grounding*: *literal.is-ground* (*literal*$_1$ $\cdot l$ $\varrho_1 \odot \gamma$)
by *simp*

have *literal*$_{G2}$*-is-strictly-maximal*$_l$:
is-strictly-maximal$_l$ (*literal.from-ground literal*$_{G2}$) (*premise*$_2$ $\cdot$ $\varrho_2 \odot \gamma$)
using *ground-superpositionI*(*11*)
unfolding *is-strictly-maximal*$_{Gl}$*-iff-is-strictly-maximal*$_l$
by (*simp add*: *premise*$_2$*-grounding*)

obtain *literal*$_2$ **where**
literal$_2$*-strictly-maximal*: *is-strictly-maximal*$_l$ *literal*$_2$ *premise*$_2$ **and**
literal$_2$*-*γ: *literal*$_2$ $\cdot l$ $\varrho_2 \odot \gamma =$ *literal.from-ground literal*$_{G2}$
using *is-strictly-maximal*$_l$*-ground-subst-stability*[*OF*
premise$_2$*-grounding*[*folded clause.subst-comp-subst*]
literal$_{G2}$*-is-strictly-maximal*$_l$
].

then have *literal*$_2$*-in-premise*$_2$: *literal*$_2$ $\in\#$ *premise*$_2$
using *strictly-maximal*$_l$*-in-clause* **by** *blast*

have *literal*$_2$*-grounding*: *literal.is-ground* (*literal*$_2$ $\cdot l$ $\varrho_2 \odot \gamma$)
using *literal*$_2$*-*γ **by** *simp*

obtain *premise*$_1{}'$ **where** *premise*$_1$: *premise*$_1$ = *add-mset literal*$_1$ *premise*$_1{}'$
by (*meson literal*$_1$*-in-premise*$_1$ *multi-member-split*)

then have *premise*$_1{}'$*-*γ: *premise*$_1{}'$ $\cdot$ ϱ_1 $\cdot$ γ = *clause.from-ground premise*$_{G1}{}'$
using *premise*$_1$*-*γ
unfolding *clause-from-ground-add-mset literal*$_1$*-*γ[*symmetric*]
by (*simp add*: *subst-clause-add-mset*)

obtain *premise*$_2{}'$ **where** *premise*$_2$: *premise*$_2$ = *add-mset literal*$_2$ *premise*$_2{}'$
by (*meson literal*$_2$*-in-premise*$_2$ *multi-member-split*)

then have *premise*$_2{}'$*-*γ: *premise*$_2{}'$ $\cdot$ ϱ_2 $\cdot$ γ = *clause.from-ground premise*$_{G2}{}'$
using *premise*$_2$*-*γ
unfolding *clause-from-ground-add-mset literal*$_2$*-*γ[*symmetric*]
by (*simp add*: *subst-clause-add-mset*)

let *?*$\mathcal{P}$ = *if* $\mathcal{P}_G$ = *Pos then Pos else Neg*

have [*simp*]: $\mathcal{P}_G \neq$ *Pos* $\longleftrightarrow$ $\mathcal{P}_G$ = *Neg*
using *ground-superpositionI*(*4*)
by *auto*

have *literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ γ =
?$\mathcal{P}$ (*Upair* (*context.from-ground context*$_G$)⟨*term.from-ground term*$_{G1}$⟩ (*term.from-ground term*$_{G2}$))
using *literal*$_1$*-*γ
unfolding *ground-superpositionI*(*5*)
by (*simp add*: *literal-from-ground-atom-from-ground atom-from-ground-term-from-ground*

ground-term-with-context(*3*))

then obtain *term*$_1$*-with-context term*$_1{}'$ **where**
literal$_1$: *literal*$_1$ = *?*$\mathcal{P}$ (*Upair term*$_1$*-with-context term*$_1{}'$) **and**
term$_1{}'$*-*γ: *term*$_1{}'$ $\cdot t$ ϱ_1 $\cdot t$ γ = *term.from-ground term*$_{G2}$ **and**

$term_1$*-with-context-*γ:
$term_1$*-with-context* $\cdot t\ \varrho_1 \cdot t\ \gamma = (context.from\text{-}ground\ context_G)\langle term.from\text{-}ground\ term_{G1}\rangle$
by (*smt* (*verit*) *obtain-from-literal-subst*)

from $literal_2$*-*γ **have** $literal_2 \cdot l\ \varrho_2 \cdot l\ \gamma = term.from\text{-}ground\ term_{G1} \approx term.from\text{-}ground\ term_{G3}$
unfolding *ground-superpositionI*(*6*) *atom-from-ground-term-from-ground literal-from-ground-atom-from-ground*(*2*) *literal.subst-comp-subst*.

then obtain $term_2\ term_2{}'$ **where**
$literal_2$: $literal_2 = term_2 \approx term_2{}'$ **and**
$term_2$*-*γ: $term_2 \cdot t\ \varrho_2 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1}$ **and**
$term_2{}'$*-*γ: $term_2{}' \cdot t\ \varrho_2 \cdot t\ \gamma = term.from\text{-}ground\ term_{G3}$
using *obtain-from-pos-literal-subst*
by *metis*

let *?inference-into-var* $= \nexists\, context_1\ term_1.$
$term_1\text{-}with\text{-}context = context_1\langle term_1\rangle \wedge$
$term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1} \wedge$
$context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G \wedge$
$is\text{-}Fun\ term_1$

have *inference-into-var-is-redundant*:
?inference-into-var $\Longrightarrow$ *ground.redundant-infer premise-groundings* ι_G
proof−
assume *inference-into-var*: *?inference-into-var*

obtain $term_x\ context_x\ context_x{}'$ **where**
$term_1$*-with-context*: $term_1\text{-}with\text{-}context = context_x\langle term_x\rangle$ **and**
is-Var-$term_x$: $is\text{-}Var\ term_x$ **and**
$context.from\text{-}ground\ context_G = (context_x \cdot t_c\ \varrho_1 \cdot t_c\ \gamma) \circ_c context_x{}'$
proof−
from *inference-into-var* $term_1$*-with-context-*γ
have
$\exists\, term_x\ context_x\ context_x{}'.$
$term_1\text{-}with\text{-}context = context_x\langle term_x\rangle \wedge$
$is\text{-}Var\ term_x \wedge$
$context.from\text{-}ground\ context_G = (context_x \cdot t_c\ \varrho_1 \cdot t_c\ \gamma) \circ_c context_x{}'$
proof(*induction* $term_1$*-with-context arbitrary*: $context_G$)
case (*Var x*)
show *?case*
proof(*intro exI conjI*)
show
$Var\ x = \Box\langle Var\ x\rangle$
$is\text{-}Var\ (Var\ x)$
$context.from\text{-}ground\ context_G = (\Box \cdot t_c\ \varrho_1 \cdot t_c\ \gamma) \circ_c context.from\text{-}ground\ context_G$
by *simp-all*

qed
next
case (*Fun f terms*)

then have $context_G \neq GHole$
by (*metis Fun.prems*(*2*) *ctxt-apply-term.simps*(*1*) *ctxt-of-gctxt.simps*(*1*)
subst-apply-ctxt.simps(*1*) *term.discI*(*2*))

then obtain $terms_{G1}$ $context_G{}'$ $terms_{G2}$ **where**
$context_G$: $context_G = GMore\ f\ terms_{G1}\ context_G{}'\ terms_{G2}$
using *Fun*(*3*)
by(*cases* $context_G$) *auto*

have *terms-γ*:
map (λ*term. term* ·*t* ϱ_1 ·*t* γ) *terms* =
map term.from-ground $terms_{G1}$ @ (*context.from-ground* $context_G{}'$)⟨*term.from-ground* $term_{G1}$⟩ #
map term.from-ground $terms_{G2}$
using *Fun*(*3*)
unfolding $context_G$
by(*simp add*: *comp-def*)

then obtain $terms_1$ *term* $terms_2$ **where**
terms: *terms* = $terms_1$ @ *term* # $terms_2$ **and**
$terms_1$-γ: *map* (λ*term. term* ·*t* ϱ_1 ·*t* γ) $terms_1$ = *map term.from-ground* $terms_{G1}$ **and**
$terms_2$-γ: *map* (λ*term. term* ·*t* ϱ_1 ·*t* γ) $terms_2$ = *map term.from-ground* $terms_{G2}$
by (*smt* (*z3*) *append-eq-map-conv map-eq-Cons-D*)

with *terms-γ*
have *term-γ*: *term* ·*t* ϱ_1 ·*t* γ = (*context.from-ground* $context_G{}'$)⟨*term.from-ground* $term_{G1}$⟩
by *simp*

show *?case*
proof(*cases term.is-ground term*)
case *True*

with *term-γ*
obtain $term_1$ $context_1$ **where**
term = $context_1$⟨$term_1$⟩
$term_1$ ·*t* ϱ_1 ·*t* γ = *term.from-ground* $term_{G1}$
$context_1$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ = *context.from-ground* $context_G{}'$
is-Fun $term_1$
by (*metis Term.ground-vars-term-empty context.ground-is-ground ground-subst-apply*
term.ground-is-ground context.subst-ident-if-ground gterm-is-fun)

moreover then have $Fun\ f\ terms = (More\ f\ terms_1\ context_1\ terms_2)\langle term_1\rangle$
  unfolding *terms*
  by *auto*

ultimately have
  $\exists\, context_1\ term_1.$
  $Fun\ f\ terms = context_1\langle term_1\rangle \wedge$
  $term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1} \wedge$
  $context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G \wedge$
  $is\text{-}Fun\ term_1$
  by (*auto*
    intro: *exI*[*of* - *More f* $terms_1\ context_1\ terms_2$] *exI*[*of* - $term_1$]
    simp: *comp-def* $terms_1\text{-}\gamma\ terms_2\text{-}\gamma\ context_G$)

then show *?thesis*
  using *Fun*(*2*)
  by *argo*
next
  case *False*
  moreover have $term \in set\ terms$
    using *terms* **by** *auto*

  moreover have
    $\nexists\, context_1\ term_1.\ term = context_1\langle term_1\rangle \wedge$
    $term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1} \wedge$
    $context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G{}' \wedge$
    $is\text{-}Fun\ term_1$
  proof(*rule notI*)
    assume
      $\exists\, context_1\ term_1.$
      $term = context_1\langle term_1\rangle \wedge$
      $term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1} \wedge$
      $context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G{}' \wedge$
      $is\text{-}Fun\ term_1$

    then obtain $context_1\ term_1$ **where**
      term: $term = context_1\langle term_1\rangle$
      $term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1}$
      $context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G{}'$
      $is\text{-}Fun\ term_1$
      by *blast*

    then have
      $\exists\, context_1\ term_1.$
      $Fun\ f\ terms = context_1\langle term_1\rangle \wedge$
      $term_1 \cdot t\ \varrho_1 \cdot t\ \gamma = term.from\text{-}ground\ term_{G1} \wedge$
      $context_1 \cdot t_c\ \varrho_1 \cdot t_c\ \gamma = context.from\text{-}ground\ context_G \wedge$
      $is\text{-}Fun\ term_1$
      by(*auto*

intro: *exI*[*of* - (*More f terms*$_1$ *context*$_1$ *terms*$_2$)] *exI*[*of* - *term*$_1$]
simp: *term terms terms*$_1$-γ *terms*$_2$-γ *context*$_G$ *comp-def*)

then show *False*
using *Fun*(*2*)
by *argo*
qed

ultimately obtain *term*$_x$ *context*$_x$ *context*$_x$*'* **where**
term = *context*$_x$⟨*term*$_x$⟩
is-Var term$_x$
context.from-ground context$_G$*'* = (*context*$_x$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ) $\circ_c$ *context*$_x$*'*
using *Fun*(*1*) *term*-γ **by** *blast*

then have
Fun f terms = (*More f terms*$_1$ *context*$_x$ *terms*$_2$)⟨*term*$_x$⟩
is-Var term$_x$
context.from-ground context$_G$ = (*More f terms*$_1$ *context*$_x$ *terms*$_2$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ) $\circ_c$ *context*$_x$*'*
by(*auto simp*: *terms terms*$_1$-γ *terms*$_2$-γ *context*$_G$ *comp-def*)

then show *?thesis*
by *blast*
qed
qed

then show *?thesis*
using *that*
by *blast*
qed

then have *context*$_G$: *context.from-ground context*$_G$ = *context*$_x$ $\circ_c$ *context*$_x$*'* $\cdot t_c$ ϱ_1 $\cdot t_c$ γ
using *ground-context-subst*[*OF context.ground-is-ground*] *ctxt-compose-subst-compose-distrib*
by *metis*

obtain τ_x **where** τ_x: *welltyped typeof-fun* $\mathcal{V}_1$ *term*$_x$ τ_x
using *term*$_1$-*with-context typing*(*1*)
unfolding *premise*$_1$ *welltyped*$_c$-*def literal*$_1$ *welltyped*$_l$-*def welltyped*$_a$-*def*
by (*metis welltyped.simps is-Var-term*$_x$ *term.collapse*(*1*))

have ι_G-*parts*:
set (*side-prems-of* ι_G) = {*premise*$_{G2}$}
main-prem-of ι_G = *premise*$_{G1}$
concl-of ι_G = *conclusion*$_G$
unfolding ι_G
by *simp-all*

from *is-Var-term*$_x$

```
obtain varₓ where varₓ: Var varₓ = termₓ ·t ϱ₁
  using renaming(1)
  unfolding is-Var-def term-subst.is-renaming-def subst-compose-def
  by (metis eval-term.simps(1) subst-apply-eq-Var)

have τₓ-varₓ: welltyped typeof-fun 𝒱₁ (Var varₓ) τₓ
  using τₓ typing(6)
  unfolding welltyped_σ-on-def varₓ premise₁ literal₁ term₁-with-context
  by(clause-auto simp: welltyped_σ-on-def welltyped_σ-on-welltyped)

show ?thesis
proof(unfold ground.redundant-infer-def ι_G-parts, intro exI conjI)

  let ?update = (contextₓ' ·t_c ϱ₁ ·t_c γ)⟨term.from-ground term_G3⟩

  define γ' where
    γ': γ' ≡ γ(varₓ := ?update)

  have update-grounding: term.is-ground ?update
  proof−
    have context.is-ground ((contextₓ ·t_c ϱ₁ ·t_c γ) ∘_c (contextₓ' ·t_c ϱ₁ ·t_c γ))
      using context.ground-is-ground[of context_G] context_G
      by fastforce

    then show ?thesis
      using context-is-ground-context-compose1(2)
      by auto
  qed
  let ?contextₓ'-γ = context.to-ground (contextₓ' ·t_c ϱ₁ ·t_c γ)

  note term-from-ground-context =
  ground-term-with-context1[OF - term.ground-is-ground, unfolded term.from-ground-inverse]

  have termₓ-γ: term.to-ground (termₓ ·t ϱ₁ ·t γ) = ?contextₓ'-γ⟨term_G1⟩_G
    using term₁-with-context-γ update-grounding
    unfolding term₁-with-context context_G
    by(auto simp: term-from-ground-context)

  have termₓ-γ': term.to-ground (termₓ ·t ϱ₁ ·t γ') = ?contextₓ'-γ⟨term_G3⟩_G
    using update-grounding
    unfolding varₓ[symmetric] γ'
    by(auto simp: term-from-ground-context)

 have aux: termₓ ·t ϱ₁ ·t γ = (contextₓ' ·t_c ϱ₁ ·t_c γ)⟨term.from-ground term_G1⟩
    using termₓ-γ
    by (metis ground-term-with-context2 term-subst.is-ground-subst-is-ground
     term-with-context-is-ground term.to-ground-inverse typing(3) update-grounding)

  have welltyped_c typeof-fun 𝒱₂ (clause.from-ground premise_G2)
```

by (*metis ground-superpositionI*(*2*) $premise_2$-γ *clause.comp-subst.left.monoid-action-compatibility typing*(*2*) *typing*(*5*) $welltyped_\sigma$*-on-*$welltyped_c$)

then have $\exists\,\tau$. *welltyped typeof-fun* $\mathcal{V}_2$ (*term.from-ground* $term_{G1}$) τ $\wedge$
welltyped typeof-fun $\mathcal{V}_2$ (*term.from-ground* $term_{G3}$) τ
unfolding *ground-superpositionI*
by *clause-simp*

then have *aux'*: $\exists\,\tau$. *welltyped typeof-fun* $\mathcal{V}_1$ (*term.from-ground* $term_{G1}$) τ $\wedge$
welltyped typeof-fun $\mathcal{V}_1$ (*term.from-ground* $term_{G3}$) τ
by (*meson term.ground-is-ground welltyped-is-ground*)

have *welltyped typeof-fun* $\mathcal{V}_1$ ($term_x$ $\cdot t$ ϱ_1 $\cdot t$ γ) τ_x
proof−

have
$[\![\forall\, x{\in}$*context.vars* $context_x$ $\cup$ *term.vars* $term_x$ $\cup$ *term.vars* $term_1{}'$ $\cup$ *clause.vars* $premise_1{}'$.
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_1$ (($\varrho_1 \odot \gamma$) x) ($\mathcal{V}_1$ x);
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_1$ $term_x$ $\tau_x]\!]$
$\Longrightarrow$ *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_1$ ($term_x$ $\cdot t$ ϱ_1 $\cdot t$ γ) τ_x
by (*metis UnI2 sup.commute term-subst.subst-comp-subst* $welltyped_\sigma$*-on-def* $welltyped_\sigma$*-on-term*)

then show *?thesis*
using *typing*(*4*) τ_x
unfolding $welltyped_\sigma$*-on-def* var_x $premise_1$ $literal_1$ $term_1$*-with-context*
by *clause-simp*
qed

then have τ_x*-update*: *welltyped typeof-fun* $\mathcal{V}_1$ *?update* τ_x
unfolding *aux*
using *aux'*
by (*meson* $welltyped_\kappa$)

let *?*$premise_1$-γ' = *clause.to-ground* ($premise_1$ $\cdot$ ϱ_1 $\cdot$ γ')
have $premise_1$-γ'*-grounding*: *clause.is-ground* ($premise_1$ $\cdot$ ϱ_1 $\cdot$ γ')
using *clause.ground-subst-upd*[*OF update-grounding* $premise_1$*-grounding*]
unfolding γ'
by *blast*

have γ'*-ground*: *term-subst.is-ground-subst* ($\varrho_1 \odot \gamma'$)
by (*metis* γ' *term.ground-subst-upd term-subst.comp-subst.left.monoid-action-compatibility*

term-subst.is-ground-subst-def typing(*3*) *update-grounding*)

have γ'*-wt*: $welltyped_\sigma$*-on* (*clause.vars* $premise_1$) *typeof-fun* $\mathcal{V}_1$ ($\varrho_1 \odot \gamma'$)

using $welltyped_\sigma$*-on-subst-upd*[*OF* τ_x*-var*$_x$ τ_x*-update typing*(*4*)]
unfolding γ' $welltyped_\sigma$*-on-def subst-compose*
using *First-Order-Type-System.welltyped.simps* τ_x τ_x*-update eval-term.simps*(*1*)

eval-with-fresh-var fun-upd-same is-Var-term$_x$ *renaming*(*1*) *subst-compose-def*

term.collapse(*1*) *term.distinct*(*1*) *term.set-cases*(*2*) *term-subst-is-renaming-iff*

the-inv-f-f typing(*4*) *var*$_x$ $welltyped_\sigma$*-on-def*
by (*smt* (*verit*, *del-insts*))

show {*?premise*$_1$-γ'} ⊆ *premise-groundings*
using *premise*$_1$-γ'*-grounding typing* γ'*-wt* γ'*-ground*
unfolding *clause.subst-comp-subst*[*symmetric*] *premise*$_1$ *premise-groundings*

clause-groundings-def
by *auto*

show *finite* {*?premise*$_1$-γ'}
by *simp*

show *ground.G-entails* ({*?premise*$_1$-γ'} ∪ {*premise*$_{G2}$}) {*conclusion*$_G$}
proof(*unfold ground.G-entails-def*, *intro allI impI*)
fix *I* :: *'f gterm rel*
let *?I* = *upair* ' *I*

assume
refl: *refl I* **and**
trans: *trans I* **and**
sym: *sym I* **and**
compatible-with-gctxt: *compatible-with-gctxt I* **and**
premise: *?I* ⊫*s* {*?premise*$_1$-γ'} ∪ {*premise*$_{G2}$}

have *var*$_x$-γ*-ground*: *term.is-ground* (*Var var*$_x$ ·*t* γ)
using *term*$_1$*-with-context*-γ
unfolding *term*$_1$*-with-context var*$_x$
by(*clause-simp simp*: *term-subst.is-ground-subst-is-ground typing*(*3*))

show *?I* ⊫*s* { *conclusion*$_G$ }
proof(*cases ?I* ⊫ *premise*$_{G2}$')
case *True*
then show *?thesis*
unfolding *ground-superpositionI*(*12*)
by *auto*
next
case *False*
then have *literal*$_{G2}$: *?I* ⊫*l literal*$_{G2}$
using *premise*
unfolding *ground-superpositionI*(*2*)

by *blast*

then have $?I \models l\ ?context_x{}'\text{-}\gamma\langle term_{G1}\rangle_G \approx ?context_x{}'\text{-}\gamma\langle term_{G3}\rangle_G$
unfolding *ground-superpositionI(6)*
using *compatible-with-gctxt compatible-with-gctxt-def sym*
by *auto*

then have $?I \models l$ *term.to-ground* $(term_x \cdot t\ \varrho_1 \cdot t\ \gamma) \approx$ *term.to-ground* $(term_x \cdot t\ \varrho_1 \cdot t\ \gamma')$
using $term_x\text{-}\gamma$ $term_x\text{-}\gamma'$
by *argo*

moreover then have $?I \models\ ?premise_1\text{-}\gamma'$
using *premise* **by** *fastforce*

ultimately have $?I \models premise_{G1}$
using
interpretation-clause-congruence[*OF*
trans sym compatible-with-gctxt update-grounding $var_x\text{-}\gamma\text{-}ground$ $premise_1\text{-}grounding$
]
var_x
unfolding γ'
by *simp*

then have $?I \models$ *add-mset* $(\mathcal{P}_G\ (Upair\ context_G\langle term_{G1}\rangle_G\ term_{G2}))\ premise_{G1}{}'$
using *ground-superpositionI(1) ground-superpositionI(5)* **by** *auto*

then have $?I \models$ *add-mset* $(\mathcal{P}_G\ (Upair\ context_G\langle term_{G3}\rangle_G\ term_{G2}))\ premise_{G1}{}'$
using
$literal_{G2}$
interpretation-context-congruence[*OF trans sym compatible-with-gctxt*]
interpretation-context-congruence'[*OF trans sym compatible-with-gctxt*]
ground-superpositionI(4)
unfolding *ground-superpositionI(6)*
by(*cases* $\mathcal{P}_G = Pos$)(*auto simp*: *sym*)

then show *?thesis*
unfolding *ground-superpositionI(12)*
by *blast*
qed
qed

show $\forall\ clause_G \in \{?premise_1\text{-}\gamma'\}.\ clause_G \prec_{cG} premise_{G1}$
proof−
have $var_x\text{-}\gamma$: $\gamma\ var_x = term_x \cdot t\ \varrho_1 \cdot t\ \gamma$
using var_x

by *simp*

have *context*$_x$*-grounding*: *context.is-ground* (*context*$_x$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ)
using *context*$_G$
unfolding *subst-compose-ctxt-compose-distrib*
by (*metis context.ground-is-ground context-is-ground-context-compose1*(*1*))

have *term*$_x$*-grounding*: *term.is-ground* (*term*$_x$ $\cdot t$ ϱ_1 $\cdot t$ γ)
using *term*$_1$*-with-context-*γ
unfolding *term*$_1$*-with-context*
by(*clause-simp simp*: *term-subst.is-ground-subst-is-ground typing*(*3*))

have
(*context*$_x$ $\circ_c$ *context*$_x$$'$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ)⟨*term.from-ground term*$_{G3}$⟩ $\prec_t$ *context*$_x$⟨*term*$_x$⟩ $\cdot t$ ϱ_1 $\cdot t$ γ
using *ground-superpositionI*(*8*)
unfolding
less$_{tG}$*-def*
context$_G$[*symmetric*]
term$_1$*-with-context*[*symmetric*]
term$_1$*-with-context-*γ
less$_t$*-ground-context-compatible-iff*[*OF*
context.ground-is-ground term.ground-is-ground term.ground-is-ground]**.**

then have *update-smaller*: *?update* $\prec_t$ γ *var*$_x$
unfolding
var$_x$*-*γ
subst-apply-term-ctxt-apply-distrib
subst-compose-ctxt-compose-distrib
Subterm-and-Context.ctxt-ctxt-compose
by(*rule less*$_t$*-ground-context-compatible*$'$[*OF*
context$_x$*-grounding update-grounding term*$_x$*-grounding*])

have *var*$_x$*-in-literal*$_1$: *var*$_x$ $\in$ *literal.vars* (*literal*$_1$ $\cdot l$ ϱ_1)
unfolding *literal*$_1$ *term*$_1$*-with-context literal.vars-def atom.vars-def*
using *var*$_x$
by(*auto simp*: *subst-literal subst-atom*)

have *literal*$_1$*-smaller*: *literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ γ' $\prec_l$ *literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ γ
unfolding γ'
using *less*$_l$*-subst-upd*[*OF*
update-grounding
update-smaller
literal$_1$*-grounding*[*unfolded literal.subst-comp-subst*]
var$_x$*-in-literal*$_1$
]**.**

have *premise*$_1$$'$*-grounding*: *clause.is-ground* (*premise*$_1$$'$ $\cdot$ ϱ_1 $\cdot$ γ)
using *premise*$_1$$'$*-*$\gamma$

by *simp*

have *premise*$_1$*'-smaller*: *premise*$_1$*'* $\cdot$ ϱ_1 $\cdot$ γ' $\preceq_c$ *premise*$_1$*'* $\cdot$ ϱ_1 $\cdot$ γ
unfolding γ'
using *less*$_c$*-subst-upd*[*of* - γ, *OF update-grounding update-smaller premise*$_1$*'-grounding*]
by(*cases var*$_x$ $\in$ *clause.vars* (*premise*$_1$*'* $\cdot$ ϱ_1)) *simp-all*

have *?premise*$_1$*-*γ' $\prec_{cG}$ *premise*$_{G1}$
using *less*$_c$*-add-mset*[*OF literal*$_1$*-smaller premise*$_1$*'-smaller*]
unfolding
less$_{cG}$*-less*$_c$
premise$_{G1}$
subst-clause-add-mset[*symmetric*]
clause.to-ground-inverse[*OF premise*$_1$*-*γ'*-grounding*]
clause.to-ground-inverse[*OF premise*$_1$*-grounding*]
unfolding *premise*$_1$.

then show *?thesis*
by *blast*
qed
qed
qed

obtain *context*$_1$ *term*$_1$ **where**
term$_1$*-with-context*: *term*$_1$*-with-context* = *context*$_1\langle$*term*$_1\rangle$ **and**
term$_1$*-*γ: *term*$_1$ $\cdot t$ ϱ_1 $\cdot t$ γ = *term.from-ground term*$_{G1}$ **and**
context$_1$*-*γ: *context*$_1$ $\cdot t_c$ ϱ_1 $\cdot t_c$ γ = *context.from-ground context*$_G$ **and**
term$_1$*-not-Var*: $\neg$ *is-Var term*$_1$
using *non-redundant ground-superposition inference-into-var-is-redundant*
unfolding
ground.Red-I-def
ground.G-Inf-def
premise-groundings
ι_G
conclusion$_G$
ground-superpositionI(*1*, *2*)
premise-groundings
by *blast*

obtain *term*$_2$*'-with-context* **where**
term$_2$*'-with-context-*γ:
term$_2$*'-with-context* $\cdot t$ γ = (*context.from-ground context*$_G$)$\langle$*term.from-ground term*$_{G3}\rangle$ **and**
term$_2$*'-with-context*: *term*$_2$*'-with-context* = (*context*$_1$ $\cdot t_c$ ϱ_1)$\langle$*term*$_2$*'* $\cdot t$ $\varrho_2\rangle$
unfolding *term*$_2$*'-*γ[*symmetric*] *context*$_1$*-*γ[*symmetric*]
by *force*

define $\mathcal{V}_3$ **where**
$\bigwedge x.\ \mathcal{V}_3\ x \equiv$

if $x \in$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1)
then $\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var x*))
else $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*))

have *wt-*γ:
welltyped$_\sigma$*-on* (*clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\cup$ *clause.vars* (*premise*$_2$ $\cdot$ ϱ_2)) *typeof-fun* $\mathcal{V}_3$ γ
proof(*unfold welltyped*$_\sigma$*-on-def*, *intro ballI*)
fix *x*
assume *x-in-vars*: $x \in$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\cup$ *clause.vars* (*premise*$_2$ $\cdot$ ϱ_2)

obtain *f ts* **where** γ*-x*: γ *x* = *Fun f ts*
using *obtain-ground-fun term-subst.is-ground-subst-is-ground*[*OF typing*(*3*)]
by (*metis eval-term.simps*(*1*))

have *welltyped typeof-fun* $\mathcal{V}_3$ (γ *x*) ($\mathcal{V}_3$ *x*)
proof(*cases* $x \in$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1))
case *True*
then have *Var x* $\in$ ϱ_1 ' *clause.vars premise*$_1$
by (*metis image-eqI renaming*(*1*) *renaming-vars-clause*)

then have *y-in-vars*: *the-inv* ϱ_1 (*Var x*) $\in$ *clause.vars premise*$_1$
by (*metis* (*no-types*, *lifting*) *image-iff renaming*(*1*) *term-subst-is-renaming-iff the-inv-f-f*)

define *y* **where** *y* $\equiv$ *the-inv* ϱ_1 (*Var x*)

have *term.is-ground* (*Var y* $\cdot t$ ϱ_1 $\cdot t$ γ)
using *term-subst.is-ground-subst-is-ground typing*(*3*) **by** *blast*

moreover have *welltyped typeof-fun* $\mathcal{V}_1$ (*Var y* $\cdot t$ ϱ_1 $\cdot t$ γ) ($\mathcal{V}_1$ *y*)
using *typing*(*4*) *y-in-vars*
unfolding *welltyped*$_\sigma$*-on-def y-def*
by (*simp add*: *subst-compose*)

ultimately have *welltyped typeof-fun* $\mathcal{V}_3$ (*Var y* $\cdot t$ ϱ_1 $\cdot t$ γ) ($\mathcal{V}_1$ *y*)
by (*meson welltyped-is-ground*)

moreover have ϱ_1 (*the-inv* ϱ_1 (*Var x*)) = *Var x*
by (*metis* ‹*Var x* $\in$ ϱ_1 ' *clause.vars premise*$_1$› *image-iff renaming*(*1*) *term-subst-is-renaming-iff the-inv-f-f*)

ultimately show *?thesis*
using *True*
unfolding $\mathcal{V}_3$*-def y-def*
by *simp*
next
case *False*
then have *Var x* $\in$ ϱ_2 ' *clause.vars premise*$_2$

using *x-in-vars*
by (*metis Un-iff image-eqI renaming*(*2*) *renaming-vars-clause*)

then have *y-in-vars*: *the-inv* ϱ_2 (*Var x*) ∈ *clause.vars premise*$_2$
by (*metis* (*no-types*, *lifting*) *image-iff renaming*(*2*) *term-subst-is-renaming-iff the-inv-f-f*)

define *y* **where** *y* ≡ *the-inv* ϱ_2 (*Var x*)

have *term.is-ground* (*Var y* ·*t* ϱ_2 ·*t* γ)
using *term-subst.is-ground-subst-is-ground typing*(*3*) **by** *blast*

moreover have *welltyped typeof-fun* $\mathcal{V}_2$ (*Var y* ·*t* ϱ_2 ·*t* γ) ($\mathcal{V}_2$ *y*)
using *typing*(*5*) *y-in-vars*
unfolding *welltyped*$_\sigma$*-on-def y-def*
by (*simp add*: *subst-compose*)

ultimately have *welltyped typeof-fun* $\mathcal{V}_3$ (*Var y* ·*t* ϱ_2 ·*t* γ) ($\mathcal{V}_2$ *y*)
by (*meson welltyped-is-ground*)

moreover have ϱ_2 (*the-inv* ϱ_2 (*Var x*)) = *Var x*
by (*metis* ‹*Var x* ∈ ϱ_2 ‘ *clause.vars premise*$_2$› *image-iff renaming*(*2*)
term-subst-is-renaming-iff the-inv-f-f)

ultimately show *?thesis*
using *False*
unfolding $\mathcal{V}_3$*-def y-def*
by *simp*
qed

then show *welltyped typeof-fun* $\mathcal{V}_3$ (γ *x*) ($\mathcal{V}_3$ *x*)
unfolding γ*-x*.
qed

have *term*$_1$ ·*t* ϱ_1 ·*t* γ = *term*$_2$ ·*t* ϱ_2 ·*t* γ
unfolding *term*$_1$-γ *term*$_2$-γ **..**

moreover have
∃τ. *welltyped typeof-fun* $\mathcal{V}_3$ (*term*$_1$ ·*t* ϱ_1) τ ∧ *welltyped typeof-fun* $\mathcal{V}_3$ (*term*$_2$ ·*t* ϱ_2) τ
proof−
have *welltyped*$_c$ *typeof-fun* $\mathcal{V}_2$ (*premise*$_2$ · ϱ_2 · γ)
using *typing*
by (*metis clause.subst-comp-subst welltyped*$_\sigma$*-on-welltyped*$_c$)

then obtain τ **where**
welltyped typeof-fun $\mathcal{V}_2$ (*term.from-ground term*$_{G1}$) τ
unfolding *premise*$_2$-γ *ground-superpositionI*
by *clause-simp*

then have
welltyped typeof-fun $\mathcal{V}_3$ (*term.from-ground* $term_{G1}$) τ
using *welltyped-is-ground*
by (*metis term.ground-is-ground*)+

then have
welltyped typeof-fun $\mathcal{V}_3$ (*term.from-ground* $term_{G1}$) τ
by *auto*

then have
welltyped typeof-fun $\mathcal{V}_3$ ($term_1 \cdot t\ \varrho_1 \cdot t\ \gamma$) τ *welltyped typeof-fun* $\mathcal{V}_3$ ($term_2 \cdot t\ \varrho_2 \cdot t\ \gamma$) τ
using $term_1$-γ $term_2$-γ
by *presburger*+

moreover have
term.vars ($term_1 \cdot t\ \varrho_1$) $\subseteq$ *clause.vars* ($premise_1 \cdot \varrho_1$)
term.vars ($term_2 \cdot t\ \varrho_2$) $\subseteq$ *clause.vars* ($premise_2 \cdot \varrho_2$)
unfolding $premise_1$ $literal_1$ *subst-clause-add-mset* $term_1$*-with-context* $premise_2$ $literal_2$
by *clause-simp*

ultimately have
welltyped typeof-fun $\mathcal{V}_3$ ($term_1 \cdot t\ \varrho_1$) τ *welltyped typeof-fun* $\mathcal{V}_3$ ($term_2 \cdot t\ \varrho_2$) τ
using *wt*-γ
unfolding $welltyped_\sigma$*-on-def*
by (*meson sup-ge1 sup-ge2* $welltyped_\sigma$*-on-subset* $welltyped_\sigma$*-on-term wt*-γ)+

then show *?thesis*
by *blast*
qed

ultimately obtain μ σ **where** μ:
term-subst.is-imgu μ $\{\{term_1 \cdot t\ \varrho_1,\ term_2 \cdot t\ \varrho_2\}\}$
$\gamma = \mu \odot \sigma$
welltyped-imgu′ typeof-fun $\mathcal{V}_3$ ($term_1 \cdot t\ \varrho_1$) ($term_2 \cdot t\ \varrho_2$) μ
using *welltyped-imgu′-exists*
by (*smt* (*verit, del-insts*))

define *conclusion′* **where**
conclusion′: *conclusion′* $\equiv$
add-mset (*?*$\mathcal{P}$ (*Upair* $term_2$*′-with-context* ($term_1' \cdot t\ \varrho_1$))) ($premise_1' \cdot \varrho_1 + premise_2' \cdot \varrho_2$) $\cdot\ \mu$

show *?thesis*
proof(*rule that*)
show *superposition* ($premise_2$, $\mathcal{V}_2$) ($premise_1$, $\mathcal{V}_1$) (*conclusion′*, $\mathcal{V}_3$)

```
proof(rule superpositionI)
  show term-subst.is-renaming ϱ₁
    using renaming(1).
next
  show term-subst.is-renaming ϱ₂
    using renaming(2).
next
  show premise₁ = add-mset literal₁ premise₁'
    using premise₁.
next
  show premise₂ = add-mset literal₂ premise₂'
    using premise₂.
next
  show ?𝒫 ∈ {Pos, Neg}
    by simp
next
  show literal₁ = ?𝒫 (Upair context₁⟨term₁⟩ term₁')
    unfolding literal₁ term₁-with-context..
next
  show literal₂ = term₂ ≈ term₂'
    using literal₂.
next
  show is-Fun term₁
    using term₁-not-Var.
next
  show term-subst.is-imgu μ {{term₁ ·t ϱ₁, term₂ ·t ϱ₂}}
    using μ(1).
next
 note premises-clause-to-ground-inverse = assms(9, 10)[THEN clause.to-ground-inverse]

 note premise-groundings = assms(10, 9)[unfolded μ(2) clause.subst-comp-subst]

  have premise₂ · ϱ₂ · μ · σ ≺c premise₁ · ϱ₁ · μ · σ
    using ground-superpositionI(3)
  unfolding premise_G1 premise_G2 less_cG-less_c premises-clause-to-ground-inverse

    unfolding μ(2) clause.subst-comp-subst
    by blast

  then show ¬ premise₁ · ϱ₁ · μ ⪯c premise₂ · ϱ₂ · μ
    using less_c-less-eq_c-ground-subst-stability[OF premise-groundings]
    by blast
next
  show ?𝒫 = Pos
        ∧ select premise₁ = {#}
        ∧ is-strictly-maximal_l (literal₁ ·l ϱ₁ ·l μ) (premise₁ · ϱ₁ · μ)
     ∨ ?𝒫 = Neg
        ∧ (select premise₁ = {#} ∧ is-maximal_l (literal₁ ·l ϱ₁ ·l μ) (premise₁ ·
ϱ₁ · μ)
```

$\vee$ *is-maximal*$_l$ (*literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ μ) ((*select premise*$_1$) $\cdot$ ϱ_1 $\cdot$ μ))
proof(*cases ?*$\mathcal{P}$ = *Pos*)
case *True*
moreover then have *select-empty*: *select premise*$_1$ = {#}
using *clause-subst-empty select(1) ground-superpositionI(9)*
by *clause-auto*

moreover have *is-strictly-maximal*$_l$ (*literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ μ $\cdot l$ σ) (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ μ $\cdot$ σ)
using *True pos-literal*$_{G1}$*-is-strictly-maximal*$_l$
unfolding *literal*$_1$-γ[*symmetric*] μ(*2*)
by *force*

moreover then have *is-strictly-maximal*$_l$ (*literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ μ) (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ μ)
using
is-strictly-maximal$_l$*-ground-subst-stability'*[*OF*
-
premise$_1$*-grounding*[*unfolded* μ(*2*) *clause.subst-comp-subst*]
]
clause.subst-in-to-set-subst
literal$_1$*-in-premise*$_1$
by *blast*

ultimately show *?thesis*
by *auto*
next
case $\mathcal{P}$*-not-Pos*: *False*
then have $\mathcal{P}_G$*-Neg*: $\mathcal{P}_G$ = *Neg*
using *ground-superpositionI(4)*
by *fastforce*

show *?thesis*
proof(*cases ?select*$_G$*-empty*)
case *select*$_G$*-empty*: *True*

then have *select premise*$_1$ = {#}
using *clause-subst-empty select(1) ground-superpositionI(9)* $\mathcal{P}_G$*-Neg*
by *clause-auto*

moreover have *is-maximal*$_l$ (*literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ μ $\cdot l$ σ) (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ μ $\cdot$ σ)
using *neg-literal*$_{G1}$*-is-maximal*$_l$[*OF select*$_G$*-empty*]
unfolding *literal*$_1$-γ[*symmetric*] μ(*2*)
by *simp*

moreover then have *is-maximal*$_l$ (*literal*$_1$ $\cdot l$ ϱ_1 $\cdot l$ μ) (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ μ)
using
is-maximal$_l$*-ground-subst-stability'*[*OF*
-

```
            premise₁-grounding[unfolded μ(2) clause.subst-comp-subst]
            ]
          clause.subst-in-to-set-subst
          literal₁-in-premise₁
        by blast

      ultimately show ?thesis
        using 𝒫_G-Neg
        by simp
    next
      case select_G-not-empty: False

      have selected-grounding: clause.is-ground (select premise₁ · ϱ₁ · μ · σ)
        using select-subst(1)[OF premise₁-grounding] select(1)
        unfolding μ(2) clause.subst-comp-subst
        by (metis clause.ground-is-ground)

      note selected-subst =
        literal₁-selected[
          OF 𝒫_G-Neg select_G-not-empty,
          THEN maximal_l-in-clause,
          THEN clause.subst-in-to-set-subst]

      have is-maximal_l (literal₁ ·l ϱ₁ ·l γ) (select premise₁ · ϱ₁ · γ)
        using select_G-not-empty ground-superpositionI(9) 𝒫_G-Neg
        unfolding is-maximal-lit-iff-is-maximal_l literal₁-γ[symmetric] select(1)
        by simp

      then have is-maximal_l (literal₁ ·l ϱ₁ ·l μ) ((select premise₁) · ϱ₁ · μ)
          using is-maximal_l-ground-subst-stability'[OF - selected-grounding] se-
lected-subst
        by (metis μ(2) clause.subst-comp-subst literal.subst-comp-subst)

      with select_G-not-empty 𝒫_G-Neg show ?thesis
        by simp
    qed
  qed
next
  show select premise₂ = {#}
    using ground-superpositionI(10) select(2)
    by clause-auto
next
  have is-strictly-maximal_l (literal₂ ·l ϱ₂ ·l μ ·l σ) (premise₂ · ϱ₂ · μ · σ)
    using literal_G2-is-strictly-maximal_l
    unfolding literal₂-γ[symmetric] μ(2)
    by simp

  then show is-strictly-maximal_l (literal₂ ·l ϱ₂ ·l μ) (premise₂ · ϱ₂ · μ)
    using
```

is-strictly-maximal$_l$*-ground-subst-stability'*[*OF*
- *premise*$_2$*-grounding*[*unfolded* μ(*2*) *clause.subst-comp-subst*]]
literal$_2$*-in-premise*$_2$
clause.subst-in-to-set-subst
by *blast*
next
have *term-groundings*:
term.is-ground (*term*$_1$*'* $\cdot t$ ϱ_1 $\cdot t$ μ $\cdot t$ σ)
term.is-ground (*context*$_1$⟨*term*$_1$⟩ $\cdot t$ ϱ_1 $\cdot t$ μ $\cdot t$ σ)
unfolding
term$_1$*-with-context*[*symmetric*]
term$_1$*-with-context-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
term$_1$*'-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
by *simp-all*

have $term_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \sigma \prec_t context_1\langle term_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \sigma$
using *ground-superpositionI*(*7*)
unfolding
term$_1$*'-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
term$_1$*-with-context*[*symmetric*]
term$_1$*-with-context-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
less$_{tG}$*-def*
ground-term-with-context(*3*).

then show $\neg\ context_1\langle term_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \preceq_t term_1' \cdot t\ \varrho_1 \cdot t\ \mu$
using *less*$_t$*-less-eq*$_t$*-ground-subst-stability*[*OF term-groundings*]
by *blast*
next
have *term-groundings*:
term.is-ground (*term*$_2$*'* $\cdot t$ ϱ_2 $\cdot t$ μ $\cdot t$ σ)
term.is-ground (*term*$_2$ $\cdot t$ ϱ_2 $\cdot t$ μ $\cdot t$ σ)
unfolding
term$_2$*-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
term$_2$*'-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
by *simp-all*

have $term_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \sigma \prec_t term_2 \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \sigma$
using *ground-superpositionI*(*8*)
unfolding
term$_2$*-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
term$_2$*'-*γ[*unfolded* μ(*2*) *term-subst.subst-comp-subst*]
less$_{tG}$*-def*.

then show $\neg\ term_2 \cdot t\ \varrho_2 \cdot t\ \mu \preceq_t term_2' \cdot t\ \varrho_2 \cdot t\ \mu$
using *less*$_t$*-less-eq*$_t$*-ground-subst-stability*[*OF term-groundings*]
by *blast*
next
show
conclusion' = *add-mset* (*?*$\mathcal{P}$ (*Upair* (*context*$_1$ $\cdot t_c$ ϱ_1)⟨*term*$_2$*'* $\cdot t$ ϱ_2⟩ (*term*$_1$*'* $\cdot t$

ϱ_1)))
$(premise_1{}' \cdot \varrho_1 + premise_2{}' \cdot \varrho_2) \cdot \mu$
unfolding *term₂'-with-context conclusion'*..
show *welltyped-imgu' typeof-fun* $\mathcal{V}_3$ $(term_1 \cdot t\ \varrho_1)$ $(term_2 \cdot t\ \varrho_2)$ μ
using $\mu(3)$ **by** *blast*

show *clause.vars* $(premise_1 \cdot \varrho_1) \cap$ *clause.vars* $(premise_2 \cdot \varrho_2) = \{\}$
using *renaming(3)*.

show $\forall x \in clause.vars\ (premise_1 \cdot \varrho_1).\ \mathcal{V}_1\ (the\text{-}inv\ \varrho_1\ (Var\ x)) = \mathcal{V}_3\ x$
unfolding $\mathcal{V}_3$*-def*
by *meson*

show $\forall x \in clause.vars\ (premise_2 \cdot \varrho_2).\ \mathcal{V}_2\ (the\text{-}inv\ \varrho_2\ (Var\ x)) = \mathcal{V}_3\ x$
unfolding $\mathcal{V}_3$*-def*
using *renaming(3)*
by (*meson disjoint-iff*)

show $welltyped_\sigma$*-on* (*clause.vars* $premise_1$) *typeof-fun* $\mathcal{V}_1$ ϱ_1
using *typing(6)*.

show $welltyped_\sigma$*-on* (*clause.vars* $premise_2$) *typeof-fun* $\mathcal{V}_2$ ϱ_2
using *typing(7)*.

have $\exists \tau.$ *welltyped typeof-fun* $\mathcal{V}_2$ $term_2$ $\tau \wedge$ *welltyped typeof-fun* $\mathcal{V}_2$ $term_2{}'$ τ
using *typing(2)*
unfolding $premise_2$ $literal_2$ $welltyped_c$*-def* $welltyped_l$*-def* $welltyped_a$*-def*
by *auto*

then show $\bigwedge\tau\ \tau'.\ [\![$*has-type typeof-fun* $\mathcal{V}_2$ $term_2$ τ; *has-type typeof-fun* $\mathcal{V}_2$ $term_2{}'\ \tau]\!] \Longrightarrow \tau = \tau'$
by (*metis welltyped-right-unique has-type-welltyped right-uniqueD*)

show *all-types* $\mathcal{V}_1$ *all-types* $\mathcal{V}_2$
using *typing*
by *auto*
qed

have *term-subst.is-idem* μ
using $\mu(1)$
by (*simp add*: *term-subst.is-imgu-iff-is-idem-and-is-mgu*)

then have μ-γ: $\mu \odot \gamma = \gamma$
unfolding $\mu(2)$ *term-subst.is-idem-def*
by (*metis subst-compose-assoc*)

have *conclusion'*-γ: *conclusion'* $\cdot\ \gamma =$ *conclusion* $\cdot\ \gamma$
proof−
have *conclusion* $\cdot\ \gamma =$

add-mset (*?*$\mathcal{P}$ (*Upair* (*context.from-ground context*$_G$)⟨*term.from-ground term*$_{G3}$⟩ (*term.from-ground term*$_{G2}$)))
(*clause.from-ground premise*$_{G1}$′ + *clause.from-ground premise*$_{G2}$′)
proof −
have ⟦
conclusion$_G$ = *add-mset* (*context*$_G$⟨*term*$_{G3}$⟩$_G$ ≈ *term*$_{G2}$) (*premise*$_{G1}$′ + *premise*$_{G2}$′);
clause.from-ground (*clause.to-ground* (*conclusion* · γ)) = *conclusion* · γ; $\mathcal{P}_G$ = *Pos*⟧
⟹ *conclusion* · γ =
add-mset
((*if Pos* = *Pos then Pos else Neg*)
(*Upair* (*term.from-ground context*$_G$⟨*term*$_{G3}$⟩$_G$) (*term.from-ground term*$_{G2}$)))
(*clause.from-ground premise*$_{G1}$′ + *clause.from-ground premise*$_{G2}$′)
by (*simp add*: *literal-from-ground-atom-from-ground*(*2*) *clause-from-ground-add-mset*

atom-from-ground-term-from-ground)

moreover have ⟦
conclusion$_G$ = *add-mset* (*context*$_G$⟨*term*$_{G3}$⟩$_G$!≈ *term*$_{G2}$) (*premise*$_{G1}$′ + *premise*$_{G2}$′);
clause.from-ground (*clause.to-ground* (*conclusion* · γ)) = *conclusion* · γ; $\mathcal{P}_G$ = *Neg*⟧
⟹ *conclusion* · γ =
add-mset
((*if Neg* = *Pos then Pos else Neg*)
(*Upair* (*term.from-ground context*$_G$⟨*term*$_{G3}$⟩$_G$) (*term.from-ground term*$_{G2}$)))
(*clause.from-ground premise*$_{G1}$′ + *clause.from-ground premise*$_{G2}$′)
by (*simp add*: *literal-from-ground-atom-from-ground*(*1*) *clause-from-ground-add-mset*

atom-from-ground-term-from-ground)

ultimately show *?thesis*
using *ground-superpositionI*(*4*, *12*) *clause.to-ground-inverse*[*OF conclusion-grounding*]
unfolding *ground-term-with-context*(*3*)
by *clause-simp*
qed

then show *?thesis*
unfolding
conclusion′
term$_2$′*-with-context-*γ[*symmetric*]
premise$_1$′*-*γ[*symmetric*]
premise$_2$′*-*γ[*symmetric*]
term$_1$′*-*γ[*symmetric*]
subst-clause-plus[*symmetric*]

subst-apply-term-ctxt-apply-distrib[*symmetric*]
subst-atom[*symmetric*]
unfolding
clause.subst-comp-subst[*symmetric*]
μ-γ
by(*simp add*: *subst-clause-add-mset subst-literal*)
qed

have *vars-conclusion'*:
clause.vars conclusion' $\subseteq$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\cup$ *clause.vars* (*premise*$_2$ $\cdot$ ϱ_2)
proof
fix x
assume $x \in$ *clause.vars conclusion'*

then consider
(*term*$_2$*'-with-context*) $x \in$ *term.vars* (*term*$_2$*'-with-context* $\cdot t$ μ)
| (*term*$_1$*'*) $x \in$ *term.vars* (*term*$_1$*'* $\cdot t$ ϱ_1 $\cdot t$ μ)
| (*premise*$_1$*'*) $x \in$ *clause.vars* (*premise*$_1$*'* $\cdot$ ϱ_1 $\cdot$ μ)
| (*premise*$_2$*'*) $x \in$ *clause.vars* (*premise*$_2$*'* $\cdot$ ϱ_2 $\cdot$ μ)
unfolding *conclusion' subst-clause-add-mset subst-clause-plus subst-literal*
by *clause-simp*

then show $x \in$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\cup$ *clause.vars* (*premise*$_2$ $\cdot$ ϱ_2)
proof(*cases*)
case *t*: *term*$_2$*'-with-context*
then show *?thesis*
using *vars-context-imgu*[*OF* μ(*1*)] *vars-term-imgu*[*OF* μ(*1*)]
unfolding *premise*$_1$ *literal*$_1$ *term*$_1$*-with-context premise*$_2$ *literal*$_2$ *term*$_2$*'-with-context*
apply *clause-simp*
by *blast*
next
case *term*$_1$*'*
then show *?thesis*
using *vars-term-imgu*[*OF* μ(*1*)]
unfolding *premise*$_1$ *subst-clause-add-mset literal*$_1$ *term*$_1$*-with-context premise*$_2$ *literal*$_2$
by *clause-simp*
next
case *premise*$_1$*'*
then show *?thesis*
using *vars-clause-imgu*[*OF* μ(*1*)]
unfolding *premise*$_1$ *subst-clause-add-mset literal*$_1$ *term*$_1$*-with-context premise*$_2$ *literal*$_2$
by *clause-simp*
next
case *premise*$_2$*'*
then show *?thesis*
using *vars-clause-imgu*[*OF* μ(*1*)]

unfolding *premise*$_1$ *subst-clause-add-mset literal*$_1$ *term*$_1$*-with-context premise*$_2$ *literal*$_2$
by *clause-simp*
qed
qed

have *surjx*: *surj* (λx. *the-inv* ϱ_2 (*Var x*))
using *surj-the-inv*[*OF renaming*(*2*)].

have *yy*:
$\bigwedge P\ Q\ b\ ty.\ \{x.\ (\mathit{if}\ b\ x\ \mathit{then}\ P\ x\ \mathit{else}\ Q\ x) = ty\ \} =$
$\{x.\ b\ x \wedge P\ x = ty\} \cup \{x.\ \neg b\ x \wedge Q\ x = ty\}$
by *auto*

have *qq*: $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) $= ty\}$
using *needed*[*OF surjx typing*(*9*)[*unfolded all-types-def*, *rule-format*]].

have *zz*:
$\bigwedge ty.$ $\{x.\ x \notin$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\wedge$ $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) $= ty\} =$
$\{x.\ \mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) $= ty\}$ $-$ $\{x.\ x \in$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1)$\}$
by *auto*

have $\bigwedge ty.$ *infinite* $\{x.\ x \notin$ *clause.vars* (*premise*$_1$ $\cdot$ ϱ_1) $\wedge$ $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) $= ty\}$
unfolding *zz*
using *qq*
by *auto*

then have *all-types-*$\mathcal{V}_3$: *all-types* $\mathcal{V}_3$
unfolding $\mathcal{V}_3$*-def all-types-def yy*
by *auto*

show $\iota_G \in$ *inference-groundings* (*Infer* [(*premise*$_2$, $\mathcal{V}_2$), (*premise*$_1$, $\mathcal{V}_1$)] (*conclusion*$'$, $\mathcal{V}_3$))
proof−
have ⟦*conclusion*$'$ $\cdot$ γ = *conclusion* $\cdot$ γ;
ground.ground-superposition (*clause.to-ground* (*premise*$_2$ $\cdot$ ϱ_2 $\cdot$ γ))
(*clause.to-ground* (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ γ)) (*clause.to-ground* (*conclusion* $\cdot$ γ));
welltyped$_\sigma$*-on* (*clause.vars conclusion*$'$) *typeof-fun* $\mathcal{V}_3$ γ; *all-types* $\mathcal{V}_3$⟧
$\Longrightarrow$ *First-Order-Type-System.welltyped*$_c$ *typeof-fun* $\mathcal{V}_3$ *conclusion*$'$
using ‹*superposition* (*premise*$_2$, $\mathcal{V}_2$) (*premise*$_1$, $\mathcal{V}_1$) (*conclusion*$'$, $\mathcal{V}_3$)›
superposition-preserves-typing typing(*1*) *typing*(*2*) **by** *blast*

then have
is-inference-grounding (*Infer* [(*premise*$_2$, $\mathcal{V}_2$), (*premise*$_1$, $\mathcal{V}_1$)] (*conclusion*$'$, $\mathcal{V}_3$)) ι_G γ ϱ_1 ϱ_2
using
conclusion$'$*-*γ *ground-superposition*
welltyped$_\sigma$*-on-subset*[*OF wt-*γ *vars-conclusion*$'$]

all-types-$\mathcal{V}_3$
unfolding *is-inference-grounding-def*
unfolding *ground.G-Inf-def* ι_G
by(*auto simp*: *typing renaming premise*$_1$*-grounding premise*$_2$*-grounding conclusion-grounding*)

then show *?thesis*
using *is-inference-grounding-inference-groundings*
by *blast*
qed

show *conclusion'* $\cdot\ \gamma$ = *conclusion* $\cdot\ \gamma$
using *conclusion'*-γ**.**
qed
qed

lemma *eq-resolution-ground-instance*:
assumes
$\iota_G \in$ *ground.eq-resolution-inferences*
$\iota_G \in$ *ground.Inf-from-q select*$_G$ ($\bigcup$(*clause-groundings typeof-fun ‘ premises*))
subst-stability-on typeof-fun premises
obtains ι **where**
$\iota \in$ *Inf-from premises*
$\iota_G \in$ *inference-groundings* ι
proof−
obtain *premise*$_G$ *conclusion*$_G$ **where**
ι_G : ι_G = *Infer* [*premise*$_G$] *conclusion*$_G$ **and**
ground-eq-resolution: *ground.ground-eq-resolution premise*$_G$ *conclusion*$_G$
using *assms*(*1*)
by *blast*

have *premise*$_G$*-in-groundings*: *premise*$_G$ $\in \bigcup$(*clause-groundings typeof-fun ‘ premises*)
using *assms*(*2*)
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by *simp*

obtain *premise conclusion* γ $\mathcal{V}$ **where**
clause.from-ground premise$_G$ = *premise* $\cdot\ \gamma$ **and**
clause.from-ground conclusion$_G$ = *conclusion* $\cdot\ \gamma$ **and**
select: *clause.from-ground* (*select*$_G$ *premise*$_G$) = *select premise* $\cdot\ \gamma$ **and**
premise-in-premises: (*premise*, $\mathcal{V}$) $\in$ *premises* **and**
typing: *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *premise*
term-subst.is-ground-subst γ
welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ
all-types $\mathcal{V}$
proof−
have *x*: $\bigwedge a\ b.$ ⟦$\bigwedge$*premise* γ *conclusion* $\mathcal{V}$.
⟦*clause.from-ground premise*$_G$ = *premise* $\cdot\ \gamma$;
clause.from-ground conclusion$_G$ = *conclusion* $\cdot\ \gamma$;

clause.from-ground (*select*$_G$ *premise*$_G$) = *select premise* · γ;
(*premise*, $\mathcal{V}$) ∈ *premises*;
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise*;
term-subst.is-ground-subst γ;
welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ; *all-types* $\mathcal{V}$⟧
⟹ *thesis*;
∀ *y*∈*premises*.
∀ *premise*$_G$∈*clause-groundings typeof-fun y*.
∃ *x*∈*premises*.
case x of
(*premise*, $\mathcal{V}$) ⇒
∃ γ. *premise* · γ = *clause.from-ground premise*$_G$ ∧
select$_G$ (*clause.to-ground* (*premise* · γ)) =
clause.to-ground (*select premise* · γ) ∧
First-Order-Type-System.welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise* ∧
welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ ∧
term-subst.is-ground-subst γ ∧ *all-types* $\mathcal{V}$;
Infer [*premise*$_G$] *conclusion*$_G$ ∈ *ground.G-Inf*; (*a*, *b*) ∈ *premises*;
premise$_G$ ∈ *clause-groundings typeof-fun* (*a*, *b*)⟧
⟹ *thesis*
by (*smt* (*verit*, *del-insts*) *case-prodE clause.ground-is-ground select-subst1*
clause.subst-ident-if-ground clause.from-ground-inverse clause.to-ground-inverse)

then show *?thesis*
using *assms*(*2*, *3*) *premise*$_G$*-in-groundings that*
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by *auto*
qed

then have
premise-grounding: *clause.is-ground* (*premise* · γ) **and**
premise$_G$: *premise*$_G$ = *clause.to-ground* (*premise* · γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* · γ) **and**
conclusion$_G$: *conclusion*$_G$ = *clause.to-ground* (*conclusion* · γ)
using *clause.ground-is-ground clause.from-ground-inverse*
by(*smt*(*verit*))+

obtain *conclusion′* **where**
eq-resolution: *eq-resolution* (*premise*, $\mathcal{V}$) (*conclusion′*, $\mathcal{V}$) **and**
ι_G: ι_G = *Infer* [*clause.to-ground* (*premise* · γ)] (*clause.to-ground* (*conclusion′* · γ)) **and**
inference-groundings: ι_G ∈ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion′*, $\mathcal{V}$)) **and**
conclusion′-conclusion: *conclusion′* · γ = *conclusion* · γ
using
eq-resolution-lifting[*OF*
premise-grounding
conclusion-grounding
select[*unfolded premise*$_G$]

ground-eq-resolution[*unfolded premise*$_G$ *conclusion*$_G$]
typing
]
unfolding *premise*$_G$ *conclusion*$_G$ ι_G
by *metis*

let *?*ι = *Infer* [(*premise*, $\mathcal{V}$)] (*conclusion*′, $\mathcal{V}$)

show *?thesis*
proof(*rule that*)
show *?*ι ∈ *Inf-from premises*
using *premise-in-premises eq-resolution*
unfolding *Inf-from-def inferences-def inference-system.Inf-from-def*
by *auto*

show ι_G ∈ *inference-groundings ?*ι
using *inference-groundings*.
qed
qed

lemma *eq-factoring-ground-instance*:
assumes
ι_G ∈ *ground.eq-factoring-inferences*
ι_G ∈ *ground.Inf-from-q select*$_G$ ($\bigcup$ (*clause-groundings typeof-fun* ‘ *premises*))
subst-stability-on typeof-fun premises
obtains ι **where**
ι ∈ *Inf-from premises*
ι_G ∈ *inference-groundings* ι
proof−
obtain *premise*$_G$ *conclusion*$_G$ **where**
ι_G : ι_G = *Infer* [*premise*$_G$] *conclusion*$_G$ **and**
ground-eq-factoring: *ground.ground-eq-factoring premise*$_G$ *conclusion*$_G$
using *assms*(*1*)
by *blast*

have *premise*$_G$*-in-groundings*: *premise*$_G$ ∈ $\bigcup$ (*clause-groundings typeof-fun* ‘ *premises*)
using *assms*(*2*)
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by *simp*

obtain *premise conclusion* γ $\mathcal{V}$ **where**
clause.from-ground premise$_G$ = *premise* · γ **and**
clause.from-ground conclusion$_G$ = *conclusion* · γ **and**
select: *clause.from-ground* (*select*$_G$ (*clause.to-ground* (*premise* · γ))) = *select premise* · γ **and**
premise-in-premises: (*premise*, $\mathcal{V}$) ∈ *premises* **and**
typing:
welltyped$_c$ *typeof-fun* $\mathcal{V}$ *premise*
term-subst.is-ground-subst γ

welltyped$_\sigma$*-on* (*clause.vars premise*) *typeof-fun* $\mathcal{V}$ γ
all-types $\mathcal{V}$
using *assms*(*2*, *3*) *premise*$_G$*-in-groundings*
unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
by (*smt* (*verit*) *clause.subst-ident-if-ground clause.ground-is-ground old.prod.case old.prod.exhaust select-subst1 clause.to-ground-inverse*)

then have
premise-grounding: *clause.is-ground* (*premise* $\cdot$ γ) **and**
premise$_G$: *premise*$_G$ = *clause.to-ground* (*premise* $\cdot$ γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* $\cdot$ γ) **and**
conclusion$_G$: *conclusion*$_G$ = *clause.to-ground* (*conclusion* $\cdot$ γ)
by (*smt*(*verit*) *clause.ground-is-ground clause.from-ground-inverse*)+

obtain *conclusion'* **where**
eq-factoring: *eq-factoring* (*premise*, $\mathcal{V}$) (*conclusion'*, $\mathcal{V}$) **and**
inference-groundings: ι_G $\in$ *inference-groundings* (*Infer* [(*premise*, $\mathcal{V}$)] (*conclusion'*, $\mathcal{V}$)) **and**
conclusion'-conclusion: *conclusion'* $\cdot$ γ = *conclusion* $\cdot$ γ
using
eq-factoring-lifting[*OF*
premise-grounding
conclusion-grounding
select
ground-eq-factoring[*unfolded premise*$_G$ *conclusion*$_G$]
]
typing
unfolding *premise*$_G$ *conclusion*$_G$ ι_G
by *metis*

let *?*ι = *Infer* [(*premise*, $\mathcal{V}$)] (*conclusion'*, $\mathcal{V}$)

show *?thesis*
proof(*rule that*)
show *?*ι $\in$ *Inf-from premises*
using *premise-in-premises eq-factoring*
unfolding *Inf-from-def inferences-def inference-system.Inf-from-def*
by *auto*

show ι_G $\in$ *inference-groundings ?*ι
using *inference-groundings*.
qed
qed

lemma *subst-compose-if*: $\sigma \odot (\lambda var.\ if\ var \in range\text{-}vars'\ \sigma\ then\ \sigma_1\ var\ else\ \sigma_2\ var) = \sigma \odot \sigma_1$
unfolding *subst-compose-def range-vars'-def*
using *term-subst-eq-conv*
by *fastforce*

lemma *subst-compose-if'*:
  assumes *range-vars'* $\sigma \cap$ *range-vars'* $\sigma' = \{\}$
  shows $\sigma \odot (\lambda var.$ *if var* $\in$ *range-vars'* σ' *then* σ_1 *var else* σ_2 *var*$) = \sigma \odot \sigma_2$
proof −
  have $\bigwedge x.\ \sigma\ x \cdot t\ (\lambda var.$ *if var* $\in$ *range-vars'* σ' *then* σ_1 *var else* σ_2 *var*$) = \sigma\ x \cdot t$ σ_2
  proof −
    fix x
    have $\bigwedge xa.\ [\![\sigma\ x = Var\ xa;\ xa \in$ *range-vars'* $\sigma']\!] \Longrightarrow \sigma_1\ xa = \sigma_2\ xa$
      by (*metis IntI assms emptyE subst-compose-def term.set-intros*(*3*)
        term-subst.comp-subst.left.right-neutral vars-term-range-vars')
    moreover have $\bigwedge x1a\ x2\ xa.$
      $[\![\sigma\ x = Fun\ x1a\ x2;\ xa \in set\ x2]\!]$
      $\Longrightarrow xa \cdot t\ (\lambda var.$ *if var* $\in$ *range-vars'* σ' *then* σ_1 *var else* σ_2 *var*$) = xa \cdot t\ \sigma_2$
       by (*smt* (*verit*, *ccfv-threshold*) *UNIV-I UN-iff assms disjoint-iff image-iff range-vars'-def*
        term.set-intros(*4*) *term-subst-eq-conv*)

    ultimately show $\sigma\ x \cdot t\ (\lambda var.$ *if var* $\in$ *range-vars'* σ' *then* σ_1 *var else* σ_2 *var*$) = \sigma\ x \cdot t\ \sigma_2$
      by(*induction* σ x) *auto*
  qed

  then show *?thesis*
    unfolding *subst-compose-def*
    by *presburger*
qed

lemma *is-ground-subst-if*:
  assumes *term-subst.is-ground-subst* γ_1 *term-subst.is-ground-subst* γ_2
  shows *term-subst.is-ground-subst* $(\lambda var.$ *if b var then* γ_1 *var else* γ_2 *var*$)$
  using *assms*
  unfolding *term-subst.is-ground-subst-def*
  by (*simp add*: *is-ground-iff*)

lemma *superposition-ground-instance*:
  assumes
    $\iota_G \in$ *ground.superposition-inferences*
    $\iota_G \in$ *ground.Inf-from-q select*$_G$ ($\bigcup$ (*clause-groundings typeof-fun ‘ premises*))
    $\iota_G \notin$ *ground.GRed-I* ($\bigcup$ (*clause-groundings typeof-fun ‘ premises*))
    subst-stability-on typeof-fun premises
  obtains ι **where**
    $\iota \in$ *Inf-from premises*
    $\iota_G \in$ *inference-groundings* ι
proof −
  obtain *premise*$_{G1}$ *premise*$_{G2}$ *conclusion*$_G$ **where**
    ι_G : ι_G = *Infer* [*premise*$_{G2}$, *premise*$_{G1}$] *conclusion*$_G$ **and**
     ground-superposition: *ground.ground-superposition premise*$_{G2}$ *premise*$_{G1}$ *con-*

clusion$_G$
  using *assms*(*1*)
  by *blast*

have
  premise$_{G1}$*-in-groundings*: *premise*$_{G1}$ $\in \bigcup$ (*clause-groundings typeof-fun* ' *premises*)
and
  premise$_{G2}$*-in-groundings*: *premise*$_{G2}$ $\in \bigcup$ (*clause-groundings typeof-fun* ' *premises*)
  using *assms*(*2*)
  unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
  by *simp-all*

obtain *premise*$_1$ $\mathcal{V}_1$ *premise*$_2$ $\mathcal{V}_2$ γ_1 γ_2 **where**
  premise$_1$-γ_1: *premise*$_1$ $\cdot$ γ_1 = *clause.from-ground premise*$_{G1}$ **and**
  premise$_2$-γ_2: *premise*$_2$ $\cdot$ γ_2 = *clause.from-ground premise*$_{G2}$ **and**
  select:
  clause.from-ground (*select*$_G$ (*clause.to-ground* (*premise*$_1$ $\cdot$ γ_1))) = *select premise*$_1$
$\cdot$ γ_1
  clause.from-ground (*select*$_G$ (*clause.to-ground* (*premise*$_2$ $\cdot$ γ_2))) = *select premise*$_2$
$\cdot$ γ_2 **and**
  premise$_1$*-in-premises*: (*premise*$_1$, $\mathcal{V}_1$) $\in$ *premises* **and**
  premise$_2$*-in-premises*: (*premise*$_2$, $\mathcal{V}_2$) $\in$ *premises* **and**
  wt:
  welltyped$_\sigma$*-on* (*clause.vars premise*$_1$) *typeof-fun* $\mathcal{V}_1$ γ_1
  welltyped$_\sigma$*-on* (*clause.vars premise*$_2$) *typeof-fun* $\mathcal{V}_2$ γ_2
  term-subst.is-ground-subst γ_1
  term-subst.is-ground-subst γ_2
  welltyped$_c$ *typeof-fun* $\mathcal{V}_1$ *premise*$_1$
  welltyped$_c$ *typeof-fun* $\mathcal{V}_2$ *premise*$_2$
  all-types $\mathcal{V}_1$
  all-types $\mathcal{V}_2$
  using *assms*(*2*, *4*) *premise*$_{G1}$*-in-groundings premise*$_{G2}$*-in-groundings*
  unfolding ι_G *ground.Inf-from-q-def ground.Inf-from-def*
  by (*smt* (*verit*, *ccfv-threshold*) *case-prod-conv clause.ground-is-ground select-subst1*

    surj-pair clause.to-ground-inverse)

obtain ϱ_1 ϱ_2 :: ('*f*, '*v*) *subst* **where**
  renaming:
  term-subst.is-renaming ϱ_1
  term-subst.is-renaming ϱ_2
  ϱ_1 ' (*clause.vars premise*$_1$) $\cap$ ϱ_2 ' (*clause.vars premise*$_2$) = {} **and**
  wt-ϱ:
  welltyped$_\sigma$*-on* (*clause.vars premise*$_1$) *typeof-fun* $\mathcal{V}_1$ ϱ_1
  welltyped$_\sigma$*-on* (*clause.vars premise*$_2$) *typeof-fun* $\mathcal{V}_2$ ϱ_2
  using *welltyped-on-renaming-exists'*[*OF* - - *wt*(*7*,*8*)[*unfolded all-types-def*, *rule-format*]]

  by (*metis clause.finite-vars*(*1*))

have *renaming-distinct*: *clause.vars* (*premise*$_1$ · ϱ_1) ∩ *clause.vars* (*premise*$_2$ · ϱ_2) = {}
  using *renaming*(*3*)
  unfolding *renaming*(*1*,*2*)[*THEN renaming-vars-clause*, *symmetric*]
  by *blast*

from *renaming* **obtain** ϱ_1-*inv* ϱ_2-*inv* **where**
  ϱ_1-*inv*: ϱ_1 ⊙ ϱ_1-*inv* = *Var* **and**
  ϱ_2-*inv*: ϱ_2 ⊙ ϱ_2-*inv* = *Var*
  unfolding *term-subst.is-renaming-def*
  by *blast*

have *select premise*$_1$ ⊆# *premise*$_1$ *select premise*$_2$ ⊆# *premise*$_2$
  by (*simp-all add*: *select-subset*)

then have *select-subset*:
  select premise$_1$ · ϱ_1 ⊆# *premise*$_1$ · ϱ_1
  select premise$_2$ · ϱ_2 ⊆# *premise*$_2$ · ϱ_2
  by (*simp-all add*: *image-mset-subseteq-mono clause.subst-def*)

define γ **where**
  γ: ⋀*var*. γ *var* ≡
    if var ∈ *clause.vars* (*premise*$_1$ · ϱ_1)
    then (ϱ_1-*inv* ⊙ γ_1) *var*
    else (ϱ_2-*inv* ⊙ γ_2) *var*

have γ_1: ∀ *x*∈ *clause.vars premise*$_1$. (ϱ_1 ⊙ γ) *x* = γ_1 *x*
proof(*intro ballI*)
  fix *x*
  assume *x-in-vars*: *x* ∈ *clause.vars premise*$_1$

  obtain *y* **where** *y*: ϱ_1 *x* = *Var y*
    by (*meson is-Var-def renaming*(*1*) *term-subst-is-renaming-iff*)

  then have *y* ∈ *clause.vars* (*premise*$_1$ · ϱ_1)
    using *x-in-vars renaming*(*1*) *renaming-vars-clause* **by** *fastforce*

  then have γ *y* = ϱ_1-*inv y* ·*t* γ_1
    by (*simp add*: γ *subst-compose*)

  then show (ϱ_1 ⊙ γ) *x* = γ_1 *x*
    by (*metis y* ϱ_1-*inv eval-term.simps*(*1*) *subst-compose*)
qed

have γ_2: ∀ *x*∈ *clause.vars premise*$_2$. (ϱ_2 ⊙ γ) *x* = γ_2 *x*
proof(*intro ballI*)
  fix *x*
  assume *x-in-vars*: *x* ∈ *clause.vars premise*$_2$

obtain y **where** y: $\varrho_2\ x = Var\ y$
  by (*meson is-Var-def renaming(2) term-subst-is-renaming-iff*)

then have $y \in clause.vars\ (premise_2 \cdot \varrho_2)$
  using *x-in-vars renaming(2) renaming-vars-clause* **by** *fastforce*

then have $\gamma\ y = \varrho_2\text{-}inv\ y \cdot t\ \gamma_2$
  using γ *renaming-distinct subst-compose* **by** *fastforce*

then show $(\varrho_2 \odot \gamma)\ x = \gamma_2\ x$
  by (*metis y* ϱ_2*-inv eval-term.simps(1) subst-compose*)
qed

have γ_1*-is-ground*: $\forall\ x{\in}clause.vars\ premise_1.\ term.is\text{-}ground\ (\gamma_1\ x)$
 by (*metis Term.term.simps(17) insert-iff is-ground-iff term-subst.is-ground-subst-def wt(3)*)

have γ_2*-is-ground*: $\forall\ x{\in}clause.vars\ premise_2.\ term.is\text{-}ground\ (\gamma_2\ x)$
 by (*metis Term.term.simps(17) insert-iff is-ground-iff term-subst.is-ground-subst-def wt(4)*)

have *wt-*γ:
  $welltyped_\sigma\text{-}on\ (clause.vars\ premise_1)\ typeof\text{-}fun\ \mathcal{V}_1\ (\varrho_1 \odot \gamma)$
  $welltyped_\sigma\text{-}on\ (clause.vars\ premise_2)\ typeof\text{-}fun\ \mathcal{V}_2\ (\varrho_2 \odot \gamma)$
  using *wt(1,2)* $welltyped_\sigma$*-on-subset* $welltyped_\sigma$*-*$welltyped_\sigma$*-on* $\gamma_1\ \gamma_2$
  unfolding $welltyped_\sigma$*-on-def*
  by *auto*

have *term-subst.is-ground-subst* $(\varrho_1\text{-}inv \odot \gamma_1)$ *term-subst.is-ground-subst* $(\varrho_2\text{-}inv \odot \gamma_2)$
  using *term-subst.is-ground-subst-comp-right wt* **by** *blast+*

then have *is-ground-subst-*γ: *term-subst.is-ground-subst* γ
  unfolding γ
  using *is-ground-subst-if*
  by *fast*

have $premise_1$*-*γ: $premise_1 \cdot \varrho_1 \cdot \gamma = clause.from\text{-}ground\ premise_{G1}$
proof −
  have $premise_1 \cdot \varrho_1 \odot (\varrho_1\text{-}inv \odot \gamma_1) = clause.from\text{-}ground\ premise_{G1}$
  by (*metis* ϱ_1*-inv* $premise_1$*-*γ_1 *subst-monoid-mult.mult.left-neutral subst-monoid-mult.mult-assoc*)

  then show *?thesis*
    using γ_1 $premise_1$*-*γ_1 *clause.subst-eq* **by** *fastforce*
qed

have $premise_2$*-*γ: $premise_2 \cdot \varrho_2 \cdot \gamma = clause.from\text{-}ground\ premise_{G2}$

proof −
have *premise*$_2$ · ϱ_2 ⊙ (ϱ_2*-inv* ⊙ γ_2) = *clause.from-ground premise*$_{G2}$
by (*metis* ϱ_2*-inv premise*$_2$-γ_2 *subst-monoid-mult.mult.left-neutral subst-monoid-mult.mult-assoc*)

then show *?thesis*
using γ_2 *premise*$_2$-γ_2 *clause.subst-eq* **by** *force*
qed

have *premise*$_1$ · ϱ_1 · γ = *premise*$_1$ · γ_1
by (*simp add*: *premise*$_1$-γ *premise*$_1$-γ_1)

moreover have *select premise*$_1$ · ϱ_1 · γ = *select premise*$_1$ · γ_1
proof−
have *clause.vars* (*select premise*$_1$ · ϱ_1) ⊆ *clause.vars* (*premise*$_1$ · ϱ_1)
using *select-subset*(*1*) *clause-submset-vars-clause-subset* **by** *blast*

then show *?thesis*
unfolding γ
by (*smt* (*verit*, *best*) ϱ_1*-inv clause.subst-eq subsetD*
clause.comp-subst.left.monoid-action-compatibility
term-subst.comp-subst.left.right-neutral)
qed

ultimately have *select*$_1$:
clause.from-ground (*select*$_G$ (*clause.to-ground* (*premise*$_1$ · ϱ_1 · γ))) = *select*
premise$_1$ · ϱ_1 · γ
using *select*(*1*)
by *argo*

have *premise*$_2$ · ϱ_2 · γ = *premise*$_2$ · γ_2
by (*simp add*: *premise*$_2$-γ *premise*$_2$-γ_2)

moreover have *select premise*$_2$ · ϱ_2 · γ = *select premise*$_2$ · γ_2
proof−
have *clause.vars* (*select premise*$_2$ · ϱ_2) ⊆ *clause.vars* (*premise*$_2$ · ϱ_2)
using *select-subset*(*2*) *clause-submset-vars-clause-subset* **by** *blast*

then show *?thesis*
unfolding γ
by (*smt* (*verit*, *best*) γ_2 γ ‹*select premise*$_2$ ⊆# *premise*$_2$› *clause-submset-vars-clause-subset*
clause.subst-eq subset-iff clause.comp-subst.left.monoid-action-compatibility)
qed

ultimately have *select*$_2$:
clause.from-ground (*select*$_G$ (*clause.to-ground* (*premise*$_2$ · ϱ_2 · γ))) = *select*
premise$_2$ · ϱ_2 · γ
using *select*(*2*)
by *argo*

obtain *conclusion* **where**
conclusion-γ: *conclusion* $\cdot$ γ = *clause.from-ground conclusion*$_G$
by (*meson clause.ground-is-ground clause.subst-ident-if-ground*)

then have
premise$_1$*-grounding*: *clause.is-ground* (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ γ) **and**
premise$_2$*-grounding*: *clause.is-ground* (*premise*$_2$ $\cdot$ ϱ_2 $\cdot$ γ) **and**
premise$_{G1}$: *premise*$_{G1}$ = *clause.to-ground* (*premise*$_1$ $\cdot$ ϱ_1 $\cdot$ γ) **and**
premise$_{G2}$: *premise*$_{G2}$ = *clause.to-ground* (*premise*$_2$ $\cdot$ ϱ_2 $\cdot$ γ) **and**
conclusion-grounding: *clause.is-ground* (*conclusion* $\cdot$ γ) **and**
conclusion$_G$: *conclusion*$_G$ = *clause.to-ground* (*conclusion* $\cdot$ γ)
by (*simp-all add*: *premise*$_1$-γ *premise*$_2$-γ)

have *clause-groundings typeof-fun* (*premise*$_1$, $\mathcal{V}_1$) $\cup$ *clause-groundings typeof-fun* (*premise*$_2$, $\mathcal{V}_2$)
$\subseteq \bigcup$ (*clause-groundings typeof-fun* ‘ *premises*)
using *premise*$_1$*-in-premises* *premise*$_2$*-in-premises* **by** *blast*

then have ι_G*-not-redunant*:
$\iota_G \notin$ *ground.GRed-I* (*clause-groundings typeof-fun* (*premise*$_1$, $\mathcal{V}_1$) $\cup$ *clause-groundings typeof-fun* (*premise*$_2$, $\mathcal{V}_2$))
using *assms*(*3*) *ground.Red-I-of-subset*
by *blast*

then obtain *conclusion'* $\mathcal{V}_3$ **where**
superposition: *superposition* (*premise*$_2$, $\mathcal{V}_2$) (*premise*$_1$, $\mathcal{V}_1$) (*conclusion'*, $\mathcal{V}_3$) **and**
inference-groundings:
$\iota_G \in$ *inference-groundings* (*Infer* [(*premise*$_2$, $\mathcal{V}_2$), (*premise*$_1$, $\mathcal{V}_1$)] (*conclusion'*, $\mathcal{V}_3$)) **and**
conclusion'-γ*-conclusion-*γ: *conclusion'* $\cdot$ γ = *conclusion* $\cdot$ γ
using
superposition-lifting[*OF*
renaming(*1*,*2*)
renaming-distinct
premise$_1$*-grounding*
premise$_2$*-grounding*
conclusion-grounding
select$_1$
select$_2$
ground-superposition[*unfolded premise*$_{G2}$ *premise*$_{G1}$ *conclusion*$_G$]
ι_G*-not-redunant*[*unfolded* ι_G *premise*$_{G2}$ *premise*$_{G1}$ *conclusion*$_G$]
wt(*5*, *6*)
is-ground-subst-γ
wt-γ
wt-ϱ
wt(*7*, *8*)
]
unfolding ι_G *conclusion*$_G$ *premise*$_{G1}$ *premise*$_{G2}$

```
    by blast

  let ?ι = Infer [(premise₂, 𝒱₂), (premise₁, 𝒱₁)] (conclusion', 𝒱₃)

  show ?thesis
  proof(rule that)
    show ?ι ∈ Inf-from premises
      using premise₁-in-premises premise₂-in-premises superposition
      unfolding Inf-from-def inferences-def inference-system.Inf-from-def
      by auto

    show ι_G ∈ inference-groundings ?ι
      using inference-groundings.
  qed
qed

lemma ground-instances:
  assumes
    ι_G ∈ ground.Inf-from-q select_G (⋃ (clause-groundings typeof-fun ‘ premises))
    ι_G ∉ ground.Red-I (⋃ (clause-groundings typeof-fun ‘ premises))
    subst-stability-on typeof-fun premises
  obtains ι where
    ι ∈ Inf-from premises
    ι_G ∈ inference-groundings ι
proof−
  have ι_G ∈ ground.superposition-inferences ∨
        ι_G ∈ ground.eq-resolution-inferences ∨
        ι_G ∈ ground.eq-factoring-inferences
    using assms(1)
    unfolding
      ground.Inf-from-q-def
      ground.Inf-from-def
      ground.G-Inf-def
      inference-system.Inf-from-def
    by fastforce

  then show ?thesis
  proof(elim disjE)
    assume ι_G ∈ ground.superposition-inferences
    then show ?thesis
      using that superposition-ground-instance assms
      by blast
  next
    assume ι_G ∈ ground.eq-resolution-inferences
    then show ?thesis
      using that eq-resolution-ground-instance assms
      by blast
  next
    assume ι_G ∈ ground.eq-factoring-inferences
```

```
    then show ?thesis
      using that eq-factoring-ground-instance assms
      by blast
  qed
qed

end

context first-order-superposition-calculus
begin

lemma overapproximation:
  obtains select_G where
    ground-Inf-overapproximated select_G premises
    is-grounding select_G
proof−
  obtain select_G where
    subst-stability: select-subst-stability-on typeof-fun select select_G premises and
    is-grounding select_G
    using obtain-subst-stable-on-select-grounding
    by blast

  then interpret grounded-first-order-superposition-calculus
    where select_G = select_G
    by unfold-locales

  have overapproximation: ground-Inf-overapproximated select_G premises
    using ground-instances[OF - - subst-stability]
    by auto

  show thesis
    using that[OF overapproximation select_G].
qed

sublocale statically-complete-calculus ⊥_F inferences entails-𝒢 Red-I-𝒢 Red-F-𝒢
proof(unfold static-empty-ord-inter-equiv-static-inter,
    rule stat-ref-comp-to-non-ground-fam-inter,
    rule ballI)
  fix select_G
  assume select_G ∈ select_Gs
  then interpret grounded-first-order-superposition-calculus
    where select_G = select_G
    by unfold-locales (simp add: select_Gs-def)

  show statically-complete-calculus
        ground.G-Bot
        ground.G-Inf
        ground.G-entails
        ground.Red-I
```

ground.Red-F
using *ground.statically-complete-calculus-axioms*.
next
fix *clauses*

have $\bigwedge$*clauses*. $\exists$ $select_G \in select_{Gs}$. *ground-Inf-overapproximated* $select_G$ *clauses*

using *overapproximation*
unfolding $select_{Gs}$*-def*
by (*smt* (*verit*, *best*) *mem-Collect-eq*)

then show *empty-ord.saturated clauses* $\Longrightarrow$
$\exists$ $select_G \in select_{Gs}$. *ground-Inf-overapproximated* $select_G$ *clauses*.
qed

end

end

8 Integration of IsaFoR Terms and the Knuth–Bendix Order

This theory implements the abstract interface for atoms and substitutions using the IsaFoR library.

theory *IsaFoR-Term-Copy*
imports
First-Order-Terms.Unification
HOL−Cardinals.Wellorder-Extension
Open-Induction.Restricted-Predicates
Knuth-Bendix-Order.KBO
begin

This part extends and integrates and the Knuth–Bendix order defined in IsaFoR.

record *'f weights* =
w :: *'f* × *nat* ⇒ *nat*
w0 :: *nat*
pr-strict :: *'f* × *nat* ⇒ *'f* × *nat* ⇒ *bool*
least :: *'f* ⇒ *bool*
scf :: *'f* × *nat* ⇒ *nat* ⇒ *nat*

class *weighted* =
fixes *weights* :: *'a weights*
assumes *weights-adm*:
admissible-kbo
(*w weights*) (*w0 weights*) (*pr-strict weights*) ((*pr-strict weights*)$^{==}$) (*least weights*) (*scf weights*)

and *pr-strict-total*: *fi* = *gj* ∨ *pr-strict weights fi gj* ∨ *pr-strict weights gj fi*
and *pr-strict-asymp*: *asymp* (*pr-strict weights*)
and *scf-ok*: *i* < *n* ⟹ *scf weights* (*f*, *n*) *i* ≤ *1*

instantiation *unit* :: *weighted* **begin**

definition *weights-unit* :: *unit weights* **where** *weights-unit* =
(|*w* = *Suc* ∘ *snd*, *w0* = *1*, *pr-strict* = λ(-, *n*) (-, *m*). *n* > *m*, *least* = λ-. *True*, *scf* = λ- -. *1*|)

instance
by (*intro-classes*, *unfold-locales*) (*auto simp*: *weights-unit-def SN-iff-wf irreflp-def intro*: *asympI intro*!: *wf-subset*[*OF wf-inv-image*[*OF wf*], *of* - *snd*])
end

global-interpretation *KBO*:
admissible-kbo
w (*weights* :: ′*f* :: *weighted weights*) *w0* (*weights* :: ′*f* :: *weighted weights*)
pr-strict weights ((*pr-strict weights*)$^{==}$) *least weights scf weights*
defines *weight* = *KBO.weight*
and *kbo* = *KBO.kbo*
by (*simp add*: *weights-adm*)

lemma *kbo-code*[*code*]: *kbo s t* =
(*let wt* = *weight t*; *ws* = *weight s in*
if vars-term-ms (*KBO.SCF t*) ⊆# *vars-term-ms* (*KBO.SCF s*) ∧ *wt* ≤ *ws*
then
(*if wt* < *ws then* (*True*, *True*)
else
(*case s of*
Var y ⇒ (*False*, *case t of Var x* ⇒ *True* | *Fun g ts* ⇒ *ts* = [] ∧ *least weights g*)
| *Fun f ss* ⇒
(*case t of*
Var x ⇒ (*True*, *True*)
| *Fun g ts* ⇒
if pr-strict weights (*f*, *length ss*) (*g*, *length ts*) *then* (*True*, *True*)
else if (*f*, *length ss*) = (*g*, *length ts*) *then lex-ext-unbounded kbo ss ts*
else (*False*, *False*))))
else (*False*, *False*))
by (*subst KBO.kbo.simps*) (*auto simp*: *Let-def split*: *term.splits*)

definition *less-kbo s t* = *fst* (*kbo t s*)

lemma *less-kbo-gtotal*: *ground s* ⟹ *ground t* ⟹ *s* = *t* ∨ *less-kbo s t* ∨ *less-kbo t s*
unfolding *less-kbo-def* **using** *KBO.S-ground-total* **by** (*metis pr-strict-total subset-UNIV*)

lemma *less-kbo-subst*:
fixes σ :: (*'f* :: *weighted*, *'v*) *subst*
shows *less-kbo s t* $\Longrightarrow$ *less-kbo* $(s \cdot \sigma)$ $(t \cdot \sigma)$
unfolding *less-kbo-def* **by** (*rule KBO.S-subst*)

lemma *wfP-less-kbo*: *wfP less-kbo*
proof −
have *SN* {(*x*, *y*). *fst* (*kbo x y*)}
using *pr-strict-asymp* **by** (*fastforce simp*: *asympI irreflp-def intro*!: *KBO.S-SN scf-ok*)
then show *?thesis*
unfolding *SN-iff-wf wfP-def* **by** (*rule wf-subset*) (*auto simp*: *less-kbo-def*)
qed

end
theory *First-Order-Superposition-Example*
imports
IsaFoR-Term-Copy
First-Order-Superposition
begin

abbreviation *trivial-select* :: (*'f*, *'v*) *select* **where**
trivial-select - ≡ {#}

abbreviation *trivial-tiebreakers* **where**
trivial-tiebreakers - - - ≡ *False*

context
assumes *ground-critical-pair-theorem*:
$\bigwedge$(*R* :: (*'f* :: *weighted*) *gterm rel*). *ground-critical-pair-theorem R*
begin

interpretation *first-order-superposition-calculus*
trivial-select :: (*'f* :: *weighted*, *'v* :: *infinite*) *select*
less-kbo
trivial-tiebreakers
λ-. ([], ())
proof(*unfold-locales*)
fix *clause* :: (*'f*, *'v*) *atom clause*

show *trivial-select clause* ⊆# *clause*
by *simp*
next
fix *clause* :: (*'f*, *'v*) *atom clause* **and** *literal*

assume *literal* ∈# *trivial-select clause*

then show *is-neg literal*
by *simp*

next
  show *transp less-kbo*
    using *KBO.S-trans*
    unfolding *transp-def less-kbo-def*
    by *blast*
next
  show *asymp less-kbo*
    using *wfP-imp-asymp wfP-less-kbo*
    by *blast*
next
  show *Wellfounded.wfp-on* $\{$*term. term.is-ground term*$\}$ *less-kbo*
    using *Wellfounded.wfp-on-subset*[*OF wfP-less-kbo subset-UNIV*] **.**
next
  show *totalp-on* $\{$*term. term.is-ground term*$\}$ *less-kbo*
    using *less-kbo-gtotal*
    unfolding *totalp-on-def Term.ground-vars-term-empty*
    by *blast*
next
  fix
    $context_G$ :: $('f,\ 'v)$ *context* **and**
    $term_{G1}$ $term_{G2}$ :: $('f,\ 'v)$ *term*

  assume *less-kbo* $term_{G1}$ $term_{G2}$

  then show *less-kbo* $context_G\langle term_{G1}\rangle$ $context_G\langle term_{G2}\rangle$
    using *KBO.S-ctxt less-kbo-def* **by** *blast*
next
  fix
    $term_1$ $term_2$ :: $('f,\ 'v)$ *term* **and**
    γ :: $('f,\ 'v)$ *subst*

  assume *less-kbo* $term_1$ $term_2$

  then show *less-kbo* $(term_1 \cdot t\ \gamma)$ $(term_2 \cdot t\ \gamma)$
    using *less-kbo-subst* **by** *blast*
next
  fix
    $term_G$ :: $('f,\ 'v)$ *term* **and**
    $context_G$ :: $('f,\ 'v)$ *context*
  assume
    term.is-ground $term_G$
    context.is-ground $context_G$
    $context_G \neq \square$

  then show *less-kbo* $term_G$ $context_G\langle term_G\rangle$
    by (*simp add*: *KBO.S-supt less-kbo-def nectxt-imp-supt-ctxt*)
next
  show $\bigwedge(R :: ('f\ gterm \times 'f\ gterm)\ set)$. *ground-critical-pair-theorem R*
    using *ground-critical-pair-theorem* **.**

next
  show $\bigwedge clause_G.\ wfP\ (\lambda\text{- -}.\ False) \wedge transp\ (\lambda\text{- -}.\ False) \wedge asymp\ (\lambda\text{- -}.\ False)$
    by (*simp add*: *asympI*)
next
  show $\bigwedge\tau.\ \exists f.\ ([], ()) = ([], \tau)$
    by *simp*
next
  show $|UNIV :: unit\ set| \leq o\ |UNIV|$
    unfolding *UNIV-unit*
    by *simp*
qed

end

end
theory *First-Order-Superposition-Soundness*
  imports *Grounded-First-Order-Superposition*

begin

8.1 Soundness

context *grounded-first-order-superposition-calculus*
begin

abbreviation $entails_F$ (**infix** $\models_F$ *50*) **where**
  $entails_F \equiv$ *lifting.entails-*$\mathcal{G}$

lemma *welltyped-extension*:
  assumes *clause.is-ground* $(C \cdot \gamma)$ $welltyped_\sigma$*-on* (*clause.vars* C) *typeof-fun* $\mathcal{V}\ \gamma$
  obtains γ'
  where
    term-subst.is-ground-subst γ'
    $welltyped_\sigma$ *typeof-fun* $\mathcal{V}\ \gamma'$
    $\forall x \in$ *clause.vars* $C.\ \gamma\ x = \gamma'\ x$
  using *assms function-symbols*
proof−
  define γ' **where** $\bigwedge x.\ \gamma'\ x \equiv$
    if $x \in$ *clause.vars* C
    then $\gamma\ x$ *else*
    Fun (*SOME* f. *typeof-fun* $f = ([], \mathcal{V}\ x)$) []

  have *term-subst.is-ground-subst* γ'
    unfolding *term-subst.is-ground-subst-def*
  proof(*intro allI*)
    fix t
    show *term.is-ground* $(t \cdot t\ \gamma')$
    proof(*induction t*)
      case (*Var x*)

then show *?case*
using *assms*(*1*)
unfolding γ'*-def* *term-subst.is-ground-subst-def* *is-ground-iff*
by(*auto simp*: *clause.variable-grounding*)
next
case *Fun*
then show *?case*
by *simp*
qed
qed

moreover have $welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ γ'
proof−
have $\bigwedge x$. ⟦∀ *x*∈*clause.vars C*. *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}$ (γ *x*) ($\mathcal{V}$ *x*);
$\bigwedge\tau$. ∃*f*. *typeof-fun f* = ([], τ); *x* ∉ *clause.vars C*⟧
⟹ *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}$
(*Fun* (*SOME f*. *typeof-fun f* = ([], $\mathcal{V}$ *x*)) []) ($\mathcal{V}$ *x*)
by (*meson First-Order-Type-System.welltyped.intros*(*2*) *list-all2-Nil someI-ex*)

then show *?thesis*
using *assms*(*2*) *function-symbols*
unfolding γ'*-def* $welltyped_\sigma$*-def* $welltyped_\sigma$*-on-def*
by *auto*
qed

moreover have ∀ *x* ∈ *clause.vars C*. γ *x* = γ' *x*
unfolding γ'*-def*
by *auto*

ultimately show *?thesis*
using *that*
by *blast*
qed

lemma *vars-subst*: ⋃ (*term.vars* ‘ ϱ ‘ *term.vars t*) = *term.vars* (*t* ·*t* ϱ)
by(*induction t*) *auto*

lemma $vars\text{-}subst_a$: ⋃ (*term.vars* ‘ ϱ ‘ *atom.vars a*) = *atom.vars* (*a* ·*a* ϱ)
using *vars-subst*
unfolding *atom.vars-def atom.subst-def*
by (*smt* (*verit*) *SUP-UNION Sup.SUP-cong UN-extend-simps*(*10*) *uprod.set-map*)

lemma $vars\text{-}subst_l$: ⋃ (*term.vars* ‘ ϱ ‘ *literal.vars l*) = *literal.vars* (*l* ·*l* ϱ)
unfolding *literal.vars-def literal.subst-def set-literal-atm-of*
by (*metis* (*no-types*, *lifting*) *UN-insert Union-image-empty literal.map-sel* $vars\text{-}subst_a$)

lemma $vars\text{-}subst_c$: ⋃ (*term.vars* ‘ ϱ ‘ *clause.vars C*) = *clause.vars* (*C* · ϱ)
using $vars\text{-}subst_l$

unfolding *clause.vars-def clause.subst-def*
by *fastforce*

lemma *eq-resolution-sound*:
assumes *step*: *eq-resolution P C*
shows $\{P\} \models_F \{C\}$
using *step*
proof (*cases P C rule*: *eq-resolution.cases*)
case (*eq-resolutionI P L P′* s_1 s_2 μ $\mathcal{V}$ *C*)

{
fix *I* :: *′f gterm rel* **and** γ :: (*′f*, *′v*) *subst*

let *?I* = *upair ‘ I*

assume
refl-I: *refl I* **and**
premise:
$\forall P_G$. ($\exists \gamma'$. P_G = *clause.to-ground* ($P \cdot \gamma'$) $\wedge$ *term-subst.is-ground-subst* γ'
$\wedge$ *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *P* $\wedge$ *welltyped*$_\sigma$*-on* (*clause.vars P*) *typeof-fun*
$\mathcal{V}$ γ')
$\longrightarrow$ *?I* $\models$ P_G **and**
grounding: *term-subst.is-ground-subst* γ **and**
wt: *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *C welltyped*$_\sigma$*-on* (*clause.vars C*) *typeof-fun* $\mathcal{V}$ γ

have *grounding′*: *clause.is-ground* ($C \cdot \gamma$)
using *grounding*
by (*simp add*: *clause.is-ground-subst-is-ground*)

obtain γ' **where**
γ': *term-subst.is-ground-subst* γ' *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ γ'
$\forall x \in$ *clause.vars C*. γ $x = \gamma'$ x
using *welltyped-extension*[*OF grounding′ wt*(*2*)].

let *?P* = *clause.to-ground* ($P \cdot \mu \cdot \gamma'$)
let *?L* = *literal.to-ground* ($L \cdot l\ \mu \cdot l\ \gamma'$)
let *?P′* = *clause.to-ground* ($P' \cdot \mu \cdot \gamma'$)
let $?s_1$ = *term.to-ground* ($s_1 \cdot t\ \mu \cdot t\ \gamma'$)
let $?s_2$ = *term.to-ground* ($s_2 \cdot t\ \mu \cdot t\ \gamma'$)

have *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ ($P' \cdot \mu$)
using *eq-resolutionI*(*8*) *wt*(*1*)
by *blast*

moreover have *welltyped-μ*: *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ μ
using *eq-resolutionI*(*6*) *wt*(*1*)
by *auto*

ultimately have *welltyped-P′*: *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *P′*

using *welltyped*$_\sigma$*-welltyped*$_c$
by *blast*

from *welltyped-*μ **have** *welltyped*$_\sigma$*-on* (*clause.vars C*) *typeof-fun* $\mathcal{V}$ ($\mu \odot \gamma'$)
using γ'(*2*)
by (*simp add*: *subst-compose-def welltyped*$_\sigma$*-def welltyped*$_\sigma$*-on-def welltyped*$_\sigma$*-welltyped*)

moreover have *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ (*add-mset* (s_1 !≈ s_2) P')
using *eq-resolutionI*(*6*) *welltyped-add-literal*[*OF welltyped-P′*] *wt*(*1*)
by *auto*

ultimately have *?I* ⊫ *?P*
using *premise*[*rule-format*, *of ?P*, *OF exI*, *of* $\mu \odot \gamma'$] γ'(*1*)
term-subst.is-ground-subst-comp-right eq-resolutionI
by (*smt* (*verit*, *ccfv-threshold*) γ'(*2*) *clause.comp-subst.left.monoid-action-compatibility*

subst-compose-def welltyped$_\sigma$*-def welltyped*$_\sigma$*-on-def welltyped*$_\sigma$*-welltyped*)

then obtain L' **where** L'*-in-P*: L' ∈# *?P* **and** *I-models-L′*: *?I* ⊫*l* L'
by (*auto simp*: *true-cls-def*)

have [*simp*]: *?P* = *add-mset ?L ?P′*
by (*simp add*: *clause.to-ground-def eq-resolutionI*(*3*) *subst-clause-add-mset*)

have [*simp*]: *?L* = (*Neg* (*Upair ?*s_1 *?*s_2))
unfolding *eq-resolutionI*(*4*) *atom.to-ground-def literal.to-ground-def*
by *clause-auto*

have [*simp*]: *?*s_1 = *?*s_2
using *term-subst.subst-imgu-eq-subst-imgu*[*OF eq-resolutionI*(*5*)] **by** *simp*

have *is-neg ?L*
by (*simp add*: *literal.to-ground-def eq-resolutionI*(*4*) *subst-literal*)

have *?I* ⊫ *clause.to-ground* ($C \cdot \gamma$)
proof(*cases* L' = *?L*)
case *True*

then have *?I* ⊫*l* (*Neg* (*atm-of ?L*))
using *I-models-L′* **by** *simp*

moreover have *atm-of* L' ∈ *?I*
using *True reflD*[*OF refl-I*, *of ?*s_1] **by** *auto*

ultimately show *?thesis*
using *True* **by** *blast*
next
case *False*
then have L' ∈# *clause.to-ground* ($P' \cdot \mu \cdot \gamma'$)

using *L′-in-P* **by** *force*

then have $L' \in\#$ *clause.to-ground* $(C \cdot \gamma')$
unfolding *eq-resolutionI*.

then show *?thesis*
using *I-models-L′*
by (*metis* $\gamma'(3)$ *clause.subst-eq true-cls-def*)
qed
}

then show *?thesis*
unfolding *ground.G-entails-def true-clss-def clause-groundings-def*
using *eq-resolutionI*(*1*, *2*) **by** *auto*
qed

lemma *eq-factoring-sound*:
assumes *step*: *eq-factoring P C*
shows $\{P\} \models_F \{C\}$
using *step*
proof (*cases P C rule*: *eq-factoring.cases*)
case (*eq-factoringI* $P\ L_1\ L_2\ P'\ s_1\ s_1'\ t_2\ t_2'\ \mu\ \mathcal{V}\ C$)

have
$\bigwedge I\ \gamma\ \mathcal{F}_G.$ ⟦
trans I;
sym I;
$\forall P_G.$ ($\exists \gamma'.$ P_G = *clause.to-ground* $(P \cdot \gamma')$ ∧ *term-subst.is-ground-subst* γ'
∧ *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *P* ∧ *welltyped*$_\sigma$*-on* (*clause.vars P*) *typeof-fun*
$\mathcal{V}\ \gamma'$)
⟶ *upair* ‘ *I* ⊫ P_G;
term-subst.is-ground-subst γ;
welltyped$_c$ *typeof-fun* $\mathcal{V}$ *C*; *welltyped*$_\sigma$*-on* (*clause.vars C*) *typeof-fun* $\mathcal{V}\ \gamma$
⟧ ⟹ *upair* ‘ *I* ⊫ *clause.to-ground* $(C \cdot \gamma)$
proof−
fix *I* :: *′f gterm rel* **and** γ :: *′v* ⇒ (*′f*, *′v*) *Term.term*

let *?I* = *upair* ‘ *I*

assume
trans-I: *trans I* **and**
sym-I: *sym I* **and**
premise:
$\forall P_G.$ ($\exists \gamma'.$ P_G = *clause.to-ground* $(P \cdot \gamma')$ ∧ *term-subst.is-ground-subst* γ'
∧ *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *P* ∧ *welltyped*$_\sigma$*-on* (*clause.vars P*) *typeof-fun*
$\mathcal{V}\ \gamma'$)
⟶ *?I* ⊫ P_G **and**
grounding: *term-subst.is-ground-subst* γ **and**
wt: *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ *C welltyped*$_\sigma$*-on* (*clause.vars C*) *typeof-fun* $\mathcal{V}\ \gamma$

obtain γ' **where**
γ': *term-subst.is-ground-subst* γ' *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ γ'
$\forall x \in$ *clause.vars* C. $\gamma\ x = \gamma'\ x$
using *welltyped-extension*
using *grounding wt*(*2*)
by (*smt* (*verit*, *ccfv-threshold*) *clause.ground-subst-iff-base-ground-subst clause.is-ground-subst-is-ground*)

let *?P* = *clause.to-ground* ($P \cdot \mu \cdot \gamma'$)
let *?P'* = *clause.to-ground* ($P' \cdot \mu \cdot \gamma'$)
let *?L*$_1$ = *literal.to-ground* ($L_1 \cdot l\ \mu \cdot l\ \gamma'$)
let *?L*$_2$ = *literal.to-ground* ($L_2 \cdot l\ \mu \cdot l\ \gamma'$)
let *?s*$_1$ = *term.to-ground* ($s_1 \cdot t\ \mu \cdot t\ \gamma'$)
let *?s*$_1{}'$ = *term.to-ground* ($s_1{}' \cdot t\ \mu \cdot t\ \gamma'$)
let *?t*$_2$ = *term.to-ground* ($t_2 \cdot t\ \mu \cdot t\ \gamma'$)
let *?t*$_2{}'$ = *term.to-ground* ($t_2{}' \cdot t\ \mu \cdot t\ \gamma'$)
let *?C* = *clause.to-ground* ($C \cdot \gamma'$)

have *wt'*:
welltyped$_c$ *typeof-fun* $\mathcal{V}$ ($P' \cdot \mu$)
welltyped$_l$ *typeof-fun* $\mathcal{V}$ ($s_1 \approx t_2{}' \cdot l\ \mu$)
welltyped$_l$ *typeof-fun* $\mathcal{V}$ ($s_1{}'\ !\!\approx t_2{}' \cdot l\ \mu$)
using *wt*(*1*)
unfolding *eq-factoringI*(*11*) *welltyped*$_c$*-add-mset subst-clause-add-mset*
by *auto*

moreover have *welltyped-*μ: *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}$ μ
using *eq-factoringI*(*10*) *wt*(*1*)
by *blast*

ultimately have *welltyped-P'*: *welltyped*$_c$ *typeof-fun* $\mathcal{V}$ P'
using *welltyped*$_\sigma$*-welltyped*$_c$
by *blast*

have *xx*: *welltyped*$_l$ *typeof-fun* $\mathcal{V}$ ($s_1 \approx t_2{}'$) *welltyped*$_l$ *typeof-fun* $\mathcal{V}$ ($s_1{}'\ !\!\approx t_2{}'$)
using *wt'*(*2*, *3*) *welltyped*$_\sigma$*-welltyped*$_l$[*OF welltyped-*μ]
by *auto*

then have *welltyped-L*$_1$: *welltyped*$_l$ *typeof-fun* $\mathcal{V}$ ($s_1 \approx s_1{}'$)
unfolding *welltyped*$_l$*-def welltyped*$_a$*-def*
using *right-uniqueD*[*OF welltyped-right-unique*]
by (*smt* (*verit*, *best*) *insert-iff set-uprod-simps literal.sel*)

have *welltyped-L*$_2$: *welltyped*$_l$ *typeof-fun* $\mathcal{V}$ ($t_2 \approx t_2{}'$)
using *xx right-uniqueD*[*OF welltyped-right-unique*] *eq-factoringI*(*10*) *wt*(*1*)
unfolding *welltyped*$_l$*-def welltyped*$_a$*-def*
by (*smt* (*verit*) *insert-iff set-uprod-simps literal.sel*(*1*))

from *welltyped-μ* **have** $\textit{welltyped}_\sigma$ *typeof-fun* $\mathcal{V}$ $(\mu \odot \gamma')$
  using *wt*(*2*) γ'
  by (*simp add*: *subst-compose-def* $\textit{welltyped}_\sigma$*-def* $\textit{welltyped}_\sigma$*-welltyped*)

moreover have $\textit{welltyped}_c$ *typeof-fun* $\mathcal{V}$ *P*
  unfolding *eq-factoringI* $\textit{welltyped}_c$*-add-mset*
  using *welltyped-P′ welltyped-*L_1 *welltyped-*L_2
  by *blast*

ultimately have *?I* ⊫ *?P*
  using
    premise[*rule-format*, *of ?P*, *OF exI*, *of* $\mu \odot \gamma'$]
    term-subst.is-ground-subst-comp-right γ'(*1*)
  by (*metis clause.subst-comp-subst* $\textit{welltyped}_\sigma$*-def* $\textit{welltyped}_\sigma$*-on-def*)

then obtain L' **where** L'*-in-P*: $L' \in\#$ *?P* **and** *I-models-L′*: *?I* ⊫*l* L'
  by (*auto simp*: *true-cls-def*)

then have s_1*-equals-*t_2: $?t_2 = ?s_1$
  using *term-subst.subst-imgu-eq-subst-imgu*[*OF eq-factoringI*(*9*)]
  by *simp*

have L_1: $?L_1 = ?s_1 \approx ?s_1'$
  unfolding *literal.to-ground-def eq-factoringI*(*4*) *atom.to-ground-def*
  by (*simp add*: *atom.subst-def subst-literal*)

have L_2: $?L_2 = ?t_2 \approx ?t_2'$
  unfolding *literal.to-ground-def eq-factoringI*(*5*) *atom.to-ground-def*
  by (*simp add*: *atom.subst-def subst-literal*)

have *C*: *?C* = *add-mset* $(?s_1 \approx ?t_2')$ (*add-mset* (*Neg* (*Upair* $?s_1'$ $?t_2'$)) *?P′*)
  unfolding *eq-factoringI*
by (*simp add*: *clause.to-ground-def literal.to-ground-def atom.subst-def subst-clause-add-mset*

    subst-literal atom.to-ground-def)

show *?I* ⊫ *clause.to-ground* $(C \cdot \gamma)$
proof(*cases* $L' = ?L_1 \vee L' = ?L_2$)
  case *True*

  then have *I* ⊫*l Pos* $(?s_1,\ ?s_1')$ ∨ *I* ⊫*l Pos* $(?s_1,\ ?t_2')$
    using *true-lit-uprod-iff-true-lit-prod*[*OF sym-I*] *I-models-L′*
    by (*metis* L_1 L_2 s_1*-equals-*t_2)

  then have *I* ⊫*l Pos* $(?s_1,\ ?t_2')$ ∨ *I* ⊫*l Neg* $(?s_1',\ ?t_2')$
    by (*meson transD trans-I true-lit-simps*(*1*) *true-lit-simps*(*2*))

  then have *?I* ⊫*l* $?s_1 \approx ?t_2'$ ∨ *?I* ⊫*l Neg* (*Upair* $?s_1'$ $?t_2'$)
    unfolding *true-lit-uprod-iff-true-lit-prod*[*OF sym-I*].

then show *?thesis*
using *clause.subst-eq* $\gamma'(3)$ *C*
by (*smt* (*verit*, *best*) *true-cls-add-mset*)
next
case *False*
then have $L' \in\#$ *?P'*
using *L'-in-P*
unfolding *eq-factoringI*
by (*simp add*: *clause.to-ground-def subst-clause-add-mset*)

then have $L' \in\#$ *clause.to-ground* $(C \cdot \gamma)$
using *clause.subst-eq* $\gamma'(3)$ *C*
by *fastforce*

then show *?thesis*
using *I-models-L'* **by** *blast*
qed
qed

then show *?thesis*
unfolding *ground.G-entails-def true-clss-def clause-groundings-def*
using *eq-factoringI*(*1*,*2*) **by** *auto*
qed

lemma *superposition-sound*:
assumes *step*: *superposition P2 P1 C*
shows $\{P1, P2\} \models_F \{C\}$
using *step*
proof (*cases P2 P1 C rule*: *superposition.cases*)
case (*superpositionI* ϱ_1 ϱ_2 P_1 P_2 L_1 $P_1{}'$ L_2 $P_2{}'$ $\mathcal{P}$ s_1 u_1 $s_1{}'$ t_2 $t_2{}'$ μ $\mathcal{V}_3$ $\mathcal{V}_1$ $\mathcal{V}_2$ C)

have
$\bigwedge I\ \gamma.\ [\![$
refl I;
trans I;
sym I;
compatible-with-gctxt I;
$\forall P_G.\ (\exists \gamma'.\ P_G =$ *clause.to-ground* $(P_1 \cdot \gamma') \wedge$ *term-subst.is-ground-subst* $\gamma' \wedge$
welltyped$_c$ *typeof-fun* $\mathcal{V}_1$ $P_1 \wedge$ *welltyped*$_\sigma$*-on* (*clause.vars* P_1) *typeof-fun* $\mathcal{V}_1$ $\gamma' \wedge$
all-types $\mathcal{V}_1) \longrightarrow$ *upair* ‘ $I \models P_G$;
$\forall P_G.\ (\exists \gamma'.\ P_G =$ *clause.to-ground* $(P_2 \cdot \gamma') \wedge$ *term-subst.is-ground-subst* $\gamma' \wedge$
welltyped$_c$ *typeof-fun* $\mathcal{V}_2$ $P_2 \wedge$ *welltyped*$_\sigma$*-on* (*clause.vars* P_2) *typeof-fun* $\mathcal{V}_2$ $\gamma' \wedge$
all-types $\mathcal{V}_2) \longrightarrow$ *upair* ‘ $I \models P_G$;

$term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma$; $welltyped_c\ typeof\text{-}fun\ \mathcal{V}_3\ C$;
$welltyped_\sigma\text{-}on\ (clause.vars\ C)\ typeof\text{-}fun\ \mathcal{V}_3\ \gamma$; $all\text{-}types\ \mathcal{V}_3$
$\rrbracket \Longrightarrow (\lambda(x,\ y).\ Upair\ x\ y)$ ‘ $I \models clause.to\text{-}ground\ (C \cdot \gamma)$
proof −
fix $I :: 'f\ gterm\ rel$ **and** $\gamma :: 'v \Rightarrow ('f,\ 'v)\ Term.term$

let $?I = (\lambda(x,\ y).\ Upair\ x\ y)$ ‘ I

assume
refl-I: *refl I* **and**
trans-I: *trans I* **and**
sym-I: *sym I* **and**
compatible-with-ground-context-I: *compatible-with-gctxt I* **and**
premise1:
$\forall P_G.\ (\exists \gamma'.\ P_G = clause.to\text{-}ground\ (P_1 \cdot \gamma') \land term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma'$
$\land\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_1\ P_1 \land welltyped_\sigma\text{-}on\ (clause.vars\ P_1)\ typeof\text{-}fun$
$\mathcal{V}_1\ \gamma'$
$\land\ all\text{-}types\ \mathcal{V}_1) \longrightarrow ?I \models P_G$ **and**
premise2:
$\forall P_G.\ (\exists \gamma'.\ P_G = clause.to\text{-}ground\ (P_2 \cdot \gamma') \land term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma'$
$\land\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_2\ P_2 \land welltyped_\sigma\text{-}on\ (clause.vars\ P_2)\ typeof\text{-}fun$
$\mathcal{V}_2\ \gamma'$
$\land\ all\text{-}types\ \mathcal{V}_2) \longrightarrow\ ?I \models P_G$ **and**
grounding: $term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_3\ C$
$welltyped_\sigma\text{-}on\ (clause.vars\ C)\ typeof\text{-}fun\ \mathcal{V}_3\ \gamma\ all\text{-}types\ \mathcal{V}_3$

have *grounding'*: $clause.is\text{-}ground\ (C \cdot \gamma)$
using *grounding*
by (*simp add*: *clause.is-ground-subst-is-ground*)

obtain γ' **where**
γ': $term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma'\ welltyped_\sigma\ typeof\text{-}fun\ \mathcal{V}_3\ \gamma'$
$\forall x \in clause.vars\ C.\ \gamma\ x = \gamma'\ x$
using *welltyped-extension*[*OF grounding' grounding*(*3*)].

let $?P_1 = clause.to\text{-}ground\ (P_1 \cdot \varrho_1 \cdot \mu \cdot \gamma')$
let $?P_2 = clause.to\text{-}ground\ (P_2 \cdot \varrho_2 \cdot \mu \cdot \gamma')$

let $?L_1 = literal.to\text{-}ground\ (L_1 \cdot l\ \varrho_1 \cdot l\ \mu \cdot l\ \gamma')$
let $?L_2 = literal.to\text{-}ground\ (L_2 \cdot l\ \varrho_2 \cdot l\ \mu \cdot l\ \gamma')$

let $?P_1' = clause.to\text{-}ground\ (P_1' \cdot \varrho_1 \cdot \mu \cdot \gamma')$
let $?P_2' = clause.to\text{-}ground\ (P_2' \cdot \varrho_2 \cdot \mu \cdot \gamma')$

let $?s_1 = context.to\text{-}ground\ (s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')$
let $?s_1' = term.to\text{-}ground\ (s_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$
let $?t_2 = term.to\text{-}ground\ (t_2 \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma')$
let $?t_2' = term.to\text{-}ground\ (t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma')$
let $?u_1 = term.to\text{-}ground\ (u_1 \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$

let $?\mathcal{P}$ = *if* $\mathcal{P}$ = *Pos then Pos else Neg*

let $?C$ = *clause.to-ground* $(C \cdot \gamma')$

have *ground-subst*:
term-subst.is-ground-subst $(\varrho_1 \odot \mu \odot \gamma')$
term-subst.is-ground-subst $(\varrho_2 \odot \mu \odot \gamma')$
term-subst.is-ground-subst $(\mu \odot \gamma')$
using *term-subst.is-ground-subst-comp-right*[*OF* $\gamma'(1)$]
by *blast*+

have *xx*: $\forall x \in$*term.vars* $(t_2 \cdot t\ \varrho_2)$. $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) = $\mathcal{V}_3\ x$
$\forall x \in$*term.vars* $(t_2{}' \cdot t\ \varrho_2)$. $\mathcal{V}_2$ (*the-inv* ϱ_2 (*Var x*)) = $\mathcal{V}_3\ x$
using *superpositionI*(*16*)
by (*simp-all add*: *clause.vars-def local.superpositionI*(*11*) *local.superpositionI*(*8*)

subst-atom subst-clause-add-mset subst-literal(*1*) *vars-atom vars-literal*(*1*))

have *wt-t*: $\exists \tau$. *welltyped typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ $\wedge$ *welltyped typeof-fun* $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu)$ τ
proof−
have $\bigwedge \tau\ \tau'$.
$[\![\bigwedge \tau\ \tau'$.
$[\![$*has-type typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ; *has-type typeof-fun* $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2)$ $\tau']\!]$ $\Longrightarrow \tau = \tau'$;
$\forall L \in \#(P_1{}' \cdot \varrho_1 + P_2{}' \cdot \varrho_2) \cdot \mu$.
$\exists \tau$. $\forall t \in$*set-uprod* (*atm-of L*). *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_3\ t\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(u_1 \cdot t\ \varrho_1)$ τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ; *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}_3\ \mu$;
$\mathcal{P}$ = *Pos*;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$
$(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu)\langle t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu\rangle$ τ';
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1 \cdot t\ \mu)$ $\tau']\!]$
$\Longrightarrow \exists \tau$. *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ $\wedge$
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu)$ τ
$\bigwedge \tau\ \tau'$.
$[\![\bigwedge \tau\ \tau'$.
$[\![$*has-type typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ; *has-type typeof-fun* $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2)$ $\tau']\!]$ $\Longrightarrow \tau = \tau'$;
$\forall L \in \#(P_1{}' \cdot \varrho_1 + P_2{}' \cdot \varrho_2) \cdot \mu$.
$\exists \tau$. $\forall t \in$*set-uprod* (*atm-of L*). *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_3\ t\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(u_1 \cdot t\ \varrho_1)$ τ;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)$ τ; *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}_3\ \mu$;
$\mathcal{P}$ = *Neg*;

First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu)\langle t_2{}' \cdot t\ \varrho_2 \cdot t$
$\mu\rangle\ \tau'$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1 \cdot t\ \mu)\ \tau'$⟧
$\Longrightarrow \exists\,\tau.$ *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)\ \tau \wedge$
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu)\ \tau$
by (*metis welltyped*$_\kappa{}'$ *welltyped*$_\sigma$*-welltyped welltyped-has-type*)+

then show *?thesis*
using *grounding*(*2*) *superpositionI*(*9*, *14*, *19*)
unfolding *superpositionI welltyped*$_c$*-def welltyped*$_l$*-def welltyped*$_a$*-def subst-clause-add-mset*
unfolding *xx*[*THEN has-type-renaming-weaker*[*OF superpositionI*(*5*)]]
by(*auto simp*: *welltyped*$_\kappa{}'$ *subst-literal subst-atom*)
qed

have *wt-*P_1: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_1$ P_1
proof−
have *xx*: $\forall\, x \in$ *clause.vars* $(P_1{}' \cdot \varrho_1)$. $\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var x*)) $= \mathcal{V}_3\ x$
using *superpositionI*(*15*)
unfolding *superpositionI subst-clause-add-mset*
by *clause-simp*

have *wt-*$P_1{}'$: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_1$ $P_1{}'$
proof−
have ⟦*welltyped*$_l$ *typeof-fun* $\mathcal{V}_3$ ($\mathcal{P}$ (*Upair* $(s_1 \cdot t_c\ \varrho_1)\langle t_2{}' \cdot t\ \varrho_2\rangle$ $(s_1{}' \cdot t\ \varrho_1)$) $\cdot l$
μ);
welltyped$_c$ *typeof-fun* $\mathcal{V}_3$ $(P_1{}' \cdot \varrho_1 \cdot \mu)$;
welltyped$_c$ *typeof-fun* $\mathcal{V}_3$ $(P_2{}' \cdot \varrho_2 \cdot \mu)$⟧
$\Longrightarrow$ *welltyped*$_c$ *typeof-fun* $\mathcal{V}_1$ $P_1{}'$
unfolding *welltyped*$_c$*-renaming-weaker*[*OF superpositionI*(*4*) *xx*]
using *superpositionI*(*14*) *welltyped*$_\sigma$*-welltyped*$_c$
by *blast*

then show *?thesis*
using *grounding*(*2*)
unfolding *superpositionI subst-clause-add-mset subst-clause-plus well-typed*$_c$*-add-mset*
welltyped$_c$*-plus*
by *auto*
qed

from *wt-t* **have** *x1*:
$\exists\,\tau.$ *welltyped typeof-fun* $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1)\langle u_1 \cdot t\ \varrho_1\rangle\ \tau \wedge$ *welltyped typeof-fun* $\mathcal{V}_3$
$(s_1{}' \cdot t\ \varrho_1)\ \tau$
proof−
have $\exists\,\tau.$ *welltyped typeof-fun* $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu)\langle t_2{}' \cdot t\ \varrho_2 \cdot t\ \mu\rangle\ \tau \wedge$
welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1 \cdot t\ \mu)\ \tau$
using *grounding*(*2*) *superpositionI*(*9*, *14*, *15*)
unfolding *superpositionI welltyped*$_c$*-def welltyped*$_l$*-def welltyped*$_a$*-def*
by *clause-auto*

then have $\exists\,\tau$. *welltyped typeof-fun* $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu)\langle u_1 \cdot t\ \varrho_1 \cdot t\ \mu\rangle\ \tau \wedge$
welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1 \cdot t\ \mu)\ \tau$
by (*meson local.superpositionI*(*14*) *welltyped*$_\kappa$ *welltyped*$_\sigma$*-welltyped wt-t*)

then show *?thesis*
by (*metis local.superpositionI*(*14*) *subst-apply-term-ctxt-apply-distrib welltyped*$_\sigma$*-welltyped*)
qed

then have $\exists\,\tau$. *welltyped typeof-fun* $\mathcal{V}_1$ $s_1\langle u_1\rangle\ \tau \wedge$ *welltyped typeof-fun* $\mathcal{V}_1$ $s_1{}'$
τ
proof−
have *x1'*: $\bigwedge\tau$. $[\![\forall\,x{\in}$*literal.vars* $((s_1 \cdot t_c\ \varrho_1)\langle u_1 \cdot t\ \varrho_1\rangle \approx s_1{}' \cdot t\ \varrho_1) \cup$ *clause.vars*
$(P_1{}' \cdot \varrho_1)$.
$\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var* x)) $= \mathcal{V}_3\ x$;
$\bigwedge t\ \mathcal{V}\ \mathcal{V}'\ \mathcal{F}\ \tau$.
$\forall\,x{\in}$*term.vars* $(t \cdot t\ \varrho_1)$. $\mathcal{V}$ (*the-inv* ϱ_1 (*Var* x)) $= \mathcal{V}'\ x \Longrightarrow$
First-Order-Type-System.welltyped $\mathcal{F}\ \mathcal{V}\ t\ \tau =$
First-Order-Type-System.welltyped $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho_1)\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1)\langle u_1 \cdot t\ \varrho_1\rangle\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1)\ \tau$; $\mathcal{P} = Pos]\!]$
$\Longrightarrow \exists\,\tau$. *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_1$ $s_1\langle u_1\rangle\ \tau \wedge$
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_1$ $s_1{}'\ \tau$
$\bigwedge\tau$. $[\![\forall\,x{\in}$*literal.vars* $((s_1 \cdot t_c\ \varrho_1)\langle u_1 \cdot t\ \varrho_1\rangle\ !{\approx}\ s_1{}' \cdot t\ \varrho_1) \cup$ *clause.vars* $(P_1{}'$
$\cdot\ \varrho_1)$.
$\mathcal{V}_1$ (*the-inv* ϱ_1 (*Var* x)) $= \mathcal{V}_3\ x$;
$\bigwedge t\ \mathcal{V}\ \mathcal{V}'\ \mathcal{F}\ \tau$.
$\forall\,x{\in}$*term.vars* $(t \cdot t\ \varrho_1)$. $\mathcal{V}$ (*the-inv* ϱ_1 (*Var* x)) $= \mathcal{V}'\ x \Longrightarrow$
First-Order-Type-System.welltyped $\mathcal{F}\ \mathcal{V}\ t\ \tau =$
First-Order-Type-System.welltyped $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho_1)\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1 \cdot t_c\ \varrho_1)\langle u_1 \cdot t\ \varrho_1\rangle\ \tau$;
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_3$ $(s_1{}' \cdot t\ \varrho_1)\ \tau$; $\mathcal{P} = Neg]\!]$
$\Longrightarrow \exists\,\tau$. *First-Order-Type-System.welltyped typeof-fun* $\mathcal{V}_1$ $s_1\langle u_1\rangle\ \tau \wedge$
First-Order-Type-System.welltyped typeof-fun $\mathcal{V}_1$ $s_1{}'\ \tau$
by *clause-simp* (*metis* (*mono-tags*) *Un-iff welltyped-renaming-weaker*[*OF*
superpositionI(*4*)]
subst-apply-term-ctxt-apply-distrib vars-term-ctxt-apply)+

with *x1* **show** *?thesis*
using *superpositionI*(*15*) *superpositionI*(*9*)
welltyped-renaming-weaker[*OF superpositionI*(*4*)]
unfolding *superpositionI subst-clause-add-mset vars-clause-add-mset*
by(*auto simp*: *welltyped*$_\kappa{}'$ *subst-literal subst-atom*)
qed

then show *?thesis*
using *grounding*(*2*) *superpositionI*(*9*, *14*) *wt-*$P_1{}'$
unfolding *superpositionI welltyped*$_c$*-def welltyped*$_l$*-def welltyped*$_a$*-def subst-clause-add-mset*

subst-clause-plus
by *auto*
qed

have *wt-*P_2: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_2$ P_2
proof−
have *xx*: $\forall\, x{\in}clause.vars\ (P_2{}' \cdot \varrho_2).\ \mathcal{V}_2\ (the\text{-}inv\ \varrho_2\ (Var\ x)) = \mathcal{V}_3\ x$
using *superpositionI*(*16*)
unfolding *superpositionI subst-clause-add-mset*
by *clause-simp*

have *wt-*$P_2{}'$: *welltyped*$_c$ *typeof-fun* $\mathcal{V}_2$ $P_2{}'$
using *grounding*(*2*)
unfolding *superpositionI subst-clause-add-mset subst-clause-plus well-typed*$_c$*-add-mset*
welltyped$_c$*-plus welltyped*$_c$*-renaming-weaker*[*OF superpositionI*(*5*) *xx*]
using *superpositionI*(*14*) *welltyped*$_\sigma$*-welltyped*$_c$ **by** *blast*

have *tt*: $\exists\,\tau.$ *welltyped typeof-fun* $\mathcal{V}_3$ $(t_2 \cdot t\ \varrho_2)\ \tau \wedge$ *welltyped typeof-fun* $\mathcal{V}_3$ $(t_2{}' \cdot t\ \varrho_2)\ \tau$
using *wt-t*
by (*meson superpositionI*(*14*) *welltyped*$_\sigma$*-welltyped*)

show *?thesis*
proof−
have $\exists\,\tau.$ *welltyped typeof-fun* $\mathcal{V}_2$ $t_2\ \tau \wedge$ *welltyped typeof-fun* $\mathcal{V}_2$ $t_2{}'\ \tau$
using *superpositionI*(*16*) *welltyped-renaming-weaker*[*OF superpositionI*(*5*)]
unfolding *superpositionI*
by (*metis* (*no-types*, *lifting*) *Un-iff subst-atom subst-clause-add-mset subst-literal*(*1*) *tt*
vars-atom vars-clause-add-mset vars-literal(*1*))

with *wt-*$P_2{}'$ **show** *?thesis*
unfolding *welltyped*$_c$*-def welltyped*$_l$*-def welltyped*$_a$*-def superpositionI*
by *auto*
qed
qed

have *wt-*μ*-*γ: *welltyped*$_\sigma$ *typeof-fun* $\mathcal{V}_3$ $(\mu \odot \gamma')$
by (*metis* γ'(*2*) *local.superpositionI*(*14*) *subst-compose-def welltyped*$_\sigma$*-def*
welltyped$_\sigma$*-welltyped*)

have *wt-*γ: *welltyped*$_\sigma$*-on* (*clause.vars* P_1) *typeof-fun* $\mathcal{V}_1$ $(\varrho_1 \odot \mu \odot \gamma')$
welltyped$_\sigma$*-on* (*clause.vars* P_2) *typeof-fun* $\mathcal{V}_2$ $(\varrho_2 \odot \mu \odot \gamma')$
using
superpositionI(*15*, *16*)
welltyped$_\sigma$*-renaming-ground-subst-weaker*[*OF superpositionI*(*4*) *wt-*μ*-*γ *superpositionI*(*17*)

ground-subst(*3*)]
welltyped$_\sigma$*-renaming-ground-subst-weaker*[*OF superpositionI*(*5*) *wt-*μ*-*γ *superpositionI*(*18*)
ground-subst(*3*)]
unfolding *vars-subst*$_c$
by (*simp-all add*: *subst-compose-assoc*)

have *?I* $\models$ *?P*$_1$
using *premise1*[*rule-format*, *of ?P*$_1$, *OF exI*, *of* $\varrho_1 \odot \mu \odot \gamma'$] *ground-subst wt-P*$_1$ *wt-*γ
superpositionI(*27*)
by *auto*

moreover have *?I* $\models$ *?P*$_2$
using *premise2*[*rule-format*, *of ?P*$_2$, *OF exI*, *of* $\varrho_2 \odot \mu \odot \gamma'$] *ground-subst wt-P*$_2$ *wt-*γ
superpositionI(*28*)
by *auto*

ultimately obtain L_1' L_2'
where
L_1'*-in-P1*: $L_1' \in\#$ *?P*$_1$ **and**
I-models-L_1': *?I* $\models l$ L_1' **and**
L_2'*-in-P2*: $L_2' \in\#$ *?P*$_2$ **and**
I-models-L_2': *?I* $\models l$ L_2'
by (*auto simp*: *true-cls-def*)

have u_1*-equals-*t_2: *?t*$_2$ = *?u*$_1$
using *term-subst.subst-imgu-eq-subst-imgu*[*OF superpositionI*(*13*)]
by *argo*

have s_1*-*u_1: *?s*$_1\langle$*?u*$_1\rangle_G$ = *term.to-ground* ($s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma'$)$\langle u_1 \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma'\rangle$
using
ground-term-with-context(*1*)[*OF*
context.is-ground-subst-is-ground
term-subst.is-ground-subst-is-ground
]
γ'(*1*)
by *auto*

have s_1*-*t_2': (*?s*$_1$)$\langle$*?t*$_2'\rangle_G$ = *term.to-ground* ($s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma'$)$\langle t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma'\rangle$
using
ground-term-with-context(*1*)[*OF*
context.is-ground-subst-is-ground
term-subst.is-ground-subst-is-ground
]
γ'(*1*)

by *auto*

have $\mathcal{P}$*-pos-or-neg*: $\mathcal{P} = Pos \vee \mathcal{P} = Neg$
using *superpositionI*(*9*) **by** *blast*

then have L_1: $?L_1 = ?\mathcal{P}$ (*Upair* $?s_1\langle ?u_1\rangle_G$ $?s_1'$)
using s_1-u_1
unfolding *superpositionI literal.to-ground-def atom.to-ground-def*
by *clause-auto*

have *literal.to-ground*
$((s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')\langle t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma'\rangle \approx s_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma') =$
term.to-ground $(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')\langle t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma'\rangle \approx$
term.to-ground $(s_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$
by (*metis atom.to-ground-def ground-atom-in-ground-literal*(*1*) *map-uprod-simps*)

moreover have *literal.to-ground*
$((s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')\langle t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma'\rangle\ !\approx s_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma') =$
term.to-ground $(s_1 \cdot t_c\ \varrho_1 \cdot t_c\ \mu \cdot t_c\ \gamma')\langle t_2' \cdot t\ \varrho_2 \cdot t\ \mu \cdot t\ \gamma'\rangle\ !\approx$
term.to-ground $(s_1' \cdot t\ \varrho_1 \cdot t\ \mu \cdot t\ \gamma')$
by (*metis atom.to-ground-def ground-atom-in-ground-literal*(*2*) *map-uprod-simps*)

ultimately have C: $?C =$ *add-mset* ($?\mathcal{P}$ (*Upair* $(?s_1)\langle ?t_2'\rangle_G$ $(?s_1')$)) $(?P_1' + ?P_2')$
using $\mathcal{P}$*-pos-or-neg*
unfolding
s_1-t_2'
superpositionI
clause.to-ground-def
subst-clause-add-mset
subst-clause-plus
by (*auto simp*: *subst-atom subst-literal*)

show $?I \models$ *clause.to-ground* $(C \cdot \gamma)$
proof (*cases* $L_1' = ?L_1$)
case L_1'*-def*: *True*
then have $?I \models l\ ?L_1$
using *superpositionI*
using *I-models-*L_1' **by** *blast*

show *?thesis*
proof (*cases* $L_2' = ?L_2$)
case L_2'*-def*: *True*

then have *ts-in-I*: $(?t_2, ?t_2') \in I$
using *I-models-*L_2' *true-lit-uprod-iff-true-lit-prod*[*OF sym-I*] *superpositionI*(*11*)
unfolding *literal.to-ground-def atom.to-ground-def*
by (*smt* (*verit*) *literal.simps*(*9*) *map-uprod-simps atom.subst-def subst-literal*

true-lit-simps(*1*))

have *?thesis* **if** $\mathcal{P}$ = *Pos*
proof −
from *that* **have** (*?s*$_1$⟨*?t*$_2$⟩$_G$, *?s*$_1$′) ∈ *I*
using *I-models-L*$_1$′ *L*$_1$′*-def L*$_1$ *true-lit-uprod-iff-true-lit-prod*[*OF sym-I*]
u$_1$*-equals-t*$_2$
unfolding *superpositionI*
by (*smt* (*verit*, *best*) *true-lit-simps*(*1*))

then have (*?s*$_1$⟨*?t*$_2$′⟩$_G$, *?s*$_1$′) ∈ *I*
using *ts-in-I compatible-with-ground-context-I refl-I sym-I trans-I*
by (*meson compatible-with-gctxtD refl-onD1 symD trans-onD*)

then have *?I* ⊨*l* *?s*$_1$⟨*?t*$_2$′⟩$_G$ ≈ *?s*$_1$′
by *blast*

then show *?thesis*
unfolding *C that*
by (*smt* (*verit*) *C* γ′(*3*) *clause.subst-eq that true-cls-def union-single-eq-member*)
qed

moreover have *?thesis* **if** $\mathcal{P}$ = *Neg*
proof −
from *that* **have** (*?s*$_1$⟨*?t*$_2$⟩$_G$, *?s*$_1$′) ∉ *I*
using *I-models-L*$_1$′ *L*$_1$′*-def L*$_1$ *true-lit-uprod-iff-true-lit-prod*[*OF sym-I*]
u$_1$*-equals-t*$_2$
unfolding *superpositionI*
by (*smt* (*verit*, *ccfv-threshold*) *literals-distinct*(*2*) *true-lit-simps*(*2*))

then have (*?s*$_1$⟨*?t*$_2$′⟩$_G$, *?s*$_1$′) ∉ *I*
using *ts-in-I compatible-with-ground-context-I trans-I*
by (*meson compatible-with-gctxtD transD*)

then have *?I* ⊨*l* *Neg* (*Upair ?s*$_1$⟨*?t*$_2$′⟩$_G$ *?s*$_1$′)
by (*meson true-lit-uprod-iff-true-lit-prod*(*2*) *sym-I true-lit-simps*(*2*))

then show *?thesis*
unfolding *C that*
by (*smt* (*verit*, *best*) *C* γ′(*3*) *calculation clause.subst-eq true-cls-def*
union-single-eq-member)
qed

ultimately show *?thesis*
using $\mathcal{P}$*-pos-or-neg* **by** *blast*
next
case *False*
then have *L*$_2$′ ∈# *?P*$_2$′

```
          using L₂'-in-P2
          unfolding superpositionI
          by (simp add: clause.to-ground-def subst-clause-add-mset)

        then have ?I ⊫ ?P₂'
          using I-models-L₂' by blast

        then show ?thesis
          unfolding superpositionI
          by (smt (verit, ccfv-SIG) C γ'(3) clause.subst-eq local.superpositionI(26)
true-cls-union
              union-mset-add-mset-left)
      qed
    next
      case False
      then have L₁' ∈# ?P₁'
        using L₁'-in-P1
        unfolding superpositionI
        by (simp add: clause.to-ground-def subst-clause-add-mset)

      then have ?I ⊫ ?P₁'
        using I-models-L₁' by blast

      then show ?thesis
        unfolding superpositionI
            by (smt (verit, best) C γ'(3) clause.subst-eq local.superpositionI(26)
true-cls-union
            union-mset-add-mset-right)
    qed
  qed

  then show ?thesis
     unfolding ground.G-entails-def clause-groundings-def true-clss-def superposi-
tionI(1−3)
    by auto
qed

end

sublocale grounded-first-order-superposition-calculus ⊆
  sound-inference-system inferences ⊥F (⊫F)
proof unfold-locales
  fix ι
  assume ι ∈ inferences
  then show set (prems-of ι) ⊫F {concl-of ι}
    using
      eq-factoring-sound
      eq-resolution-sound
      superposition-sound
```

unfolding *inferences-def ground.G-entails-def*
by *auto*
qed

sublocale *first-order-superposition-calculus* $\subseteq$
sound-inference-system inferences $\bot_F$ *entails-*$\mathcal{G}$
proof *unfold-locales*
obtain $select_G$ **where** $select_G$: $select_G \in select_{Gs}$
using *Q-nonempty* **by** *blast*

then interpret *grounded-first-order-superposition-calculus*
where $select_G = select_G$
by *unfold-locales* (*simp add*: $select_{Gs}$*-def*)

show $\bigwedge\iota.\ \iota \in$ *inferences* $\Longrightarrow$ *entails-*$\mathcal{G}$ (*set* (*prems-of* ι)) {*concl-of* ι}
using *sound*
unfolding *entails-def*
by *blast*
qed

end
theory *Ground-Superposition-Soundness*
imports *Ground-Superposition*
begin

lemma (**in** *ground-superposition-calculus*) *soundness-ground-superposition*:
assumes
step: *ground-superposition P1 P2 C*
shows *G-entails* {*P1*, *P2*} {*C*}
using *step*
proof (*cases P1 P2 C rule*: *ground-superposition.cases*)
case (*ground-superpositionI* L_1 P_1' L_2 P_2' $\mathcal{P}$ *s t s' t'*)

show *?thesis*
unfolding *G-entails-def true-clss-singleton*
unfolding *true-clss-insert*
proof (*intro allI impI*, *elim conjE*)
fix *I* :: *'f gterm rel*
let *?I'* = $(\lambda(t_1, t).$ *Upair* t_1 $t)$ ' *I*
assume *refl I* **and** *trans I* **and** *sym I* **and** *compatible-with-gctxt I* **and**
?I' $\models$ *P1* **and** *?I'* $\models$ *P2*
then obtain *K1 K2* :: *'f gatom literal* **where**
K1 $\in\#$ *P1* **and** *?I'* $\models l$ *K1* **and** *K2* $\in\#$ *P2* **and** *?I'* $\models l$ *K2*
by (*auto simp*: *true-cls-def*)

show *?I'* $\models$ *C*
proof (*cases K2* = $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ *s'*))
case *K1-def*: *True*
hence *?I'* $\models l$ $\mathcal{P}$ (*Upair* $s\langle t\rangle_G$ *s'*)

using ‹*?I'* $\Vert$=*l K2*› **by** *simp*

show *?thesis*
proof (*cases K1* = *Pos* (*Upair t t'*))
case *K2-def*: *True*
hence (*t*, *t'*) ∈ *I*
using ‹*?I'* $\Vert$=*l K1*› *true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*

have *?thesis* **if** $\mathcal{P}$ = *Pos*
proof −
from *that* **have** ($s\langle t\rangle_G$, *s'*) ∈ *I*
using ‹*?I'* $\Vert$=*l K2*› *K1-def true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*
hence ($s\langle t'\rangle_G$, *s'*) ∈ *I*
using ‹(*t*, *t'*) ∈ *I*›
using ‹*compatible-with-gctxt I*› ‹*refl I*› ‹*sym I*› ‹*trans I*›
by (*meson compatible-with-gctxtD refl-onD1 symD trans-onD*)
hence *?I'* $\Vert$=*l Pos* (*Upair* $s\langle t'\rangle_G$ *s'*)
by *blast*
thus *?thesis*
unfolding *ground-superpositionI that*
by *simp*
qed

moreover have *?thesis* **if** $\mathcal{P}$ = *Neg*
proof −
from *that* **have** ($s\langle t\rangle_G$, *s'*) ∉ *I*
using ‹*?I'* $\Vert$=*l K2*› *K1-def true-lit-uprod-iff-true-lit-prod*[*OF* ‹*sym I*›] **by** *simp*
hence ($s\langle t'\rangle_G$, *s'*) ∉ *I*
using ‹(*t*, *t'*) ∈ *I*›
using ‹*compatible-with-gctxt I*› ‹*trans I*›
by (*metis compatible-with-gctxtD transD*)
hence *?I'* $\Vert$=*l Neg* (*Upair* $s\langle t'\rangle_G$ *s'*)
by (*meson* ‹*sym I*› *true-lit-simps*(*2*) *true-lit-uprod-iff-true-lit-prod*(*2*))
thus *?thesis*
unfolding *ground-superpositionI that* **by** *simp*
qed

ultimately show *?thesis*
using ‹$\mathcal{P}$ ∈ {*Pos*, *Neg*}› **by** *auto*
next
case *False*
hence *K1* ∈# P_2'
using ‹*K1* ∈# *P1*›
unfolding *ground-superpositionI* **by** *simp*
hence *?I'* $\Vert$= P_2'
using ‹*?I'* $\Vert$=*l K1*› **by** *blast*
thus *?thesis*

unfolding *ground-superpositionI* **by** *simp*
qed
next
case *False*
hence $K2 \in\# P_1{}'$
using ‹$K2 \in\# P2$›
unfolding *ground-superpositionI* **by** *simp*
hence $?I' \models P_1{}'$
using ‹$?I' \models l\ K2$› **by** *blast*
thus *?thesis*
unfolding *ground-superpositionI* **by** *simp*
qed
qed
qed

lemma (**in** *ground-superposition-calculus*) *soundness-ground-eq-resolution*:
assumes *step*: *ground-eq-resolution P C*
shows *G-entails* $\{P\}$ $\{C\}$
using *step*
proof (*cases P C rule*: *ground-eq-resolution.cases*)
case (*ground-eq-resolutionI L D′ t*)
show *?thesis*
unfolding *G-entails-def true-clss-singleton*
proof (*intro allI impI*)
fix *I* :: *′f gterm rel*
assume *refl I* **and** $(\lambda(t_1, t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models P$
then obtain *K* **where** $K \in\# P$ **and** $(\lambda(t_1, t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models l\ K$
by (*auto simp*: *true-cls-def*)
hence $K \neq L$
by (*metis* ‹*refl I*› *ground-eq-resolutionI*(*2*) *pair-imageI reflD true-lit-simps*(*2*))
hence $K \in\# C$
using ‹$K \in\# P$› ‹$P = add\text{-}mset\ L\ D'$› ‹$C = D'$› **by** *simp*
thus $(\lambda(t_1, t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models C$
using ‹$(\lambda(t_1, t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models l\ K$› **by** *blast*
qed
qed

lemma (**in** *ground-superposition-calculus*) *soundness-ground-eq-factoring*:
assumes *step*: *ground-eq-factoring P C*
shows *G-entails* $\{P\}$ $\{C\}$
using *step*
proof (*cases P C rule*: *ground-eq-factoring.cases*)
case (*ground-eq-factoringI* L_1 L_2 P' t t' t'')
show *?thesis*
unfolding *G-entails-def true-clss-singleton*
proof (*intro allI impI*)
fix *I* :: *′f gterm rel*
let $?I' = (\lambda(t_1, t).\ Upair\ t_1\ t)$ ‘ I
assume *trans I* **and** *sym I* **and** $?I' \models P$

```
    then obtain K :: 'f gatom literal where
      K ∈# P and ?I' ⊫l K
      by (auto simp: true-cls-def)

    show ?I' ⊫ C
    proof (cases K = L₁ ∨ K = L₂)
      case True
      hence I ⊫l Pos (t, t') ∨ I ⊫l Pos (t, t'')
        unfolding ground-eq-factoringI
        using ‹?I' ⊫l K› true-lit-uprod-iff-true-lit-prod[OF ‹sym I›] by metis
      hence I ⊫l Pos (t, t'') ∨ I ⊫l Neg (t', t'')
      proof (elim disjE)
        assume I ⊫l Pos (t, t')
        then show ?thesis
          unfolding true-lit-simps
          by (metis ‹trans I› transD)
      next
        assume I ⊫l Pos (t, t'')
        then show ?thesis
          by simp
      qed
      hence ?I' ⊫l Pos (Upair t t'') ∨ ?I' ⊫l Neg (Upair t' t'')
        unfolding true-lit-uprod-iff-true-lit-prod[OF ‹sym I›] .
      thus ?thesis
        unfolding ground-eq-factoringI
        by (metis true-cls-add-mset)
    next
      case False
      hence K ∈# P'
        using ‹K ∈# P›
        unfolding ground-eq-factoringI
        by auto
      hence K ∈# C
        by (simp add: ground-eq-factoringI(1,2,7))
      thus ?thesis
        using ‹(λ(t₁, t). Upair t₁ t) ` I ⊫l K› by blast
    qed
  qed
qed

sublocale ground-superposition-calculus ⊆ sound-inference-system where
  Inf = G-Inf and
  Bot = G-Bot and
  entails = G-entails
proof unfold-locales
  show ⋀ι. ι ∈ G-Inf ⟹ G-entails (set (prems-of ι)) {concl-of ι}
    unfolding G-Inf-def
    using soundness-ground-superposition
    using soundness-ground-eq-resolution
```

using *soundness-ground-eq-factoring*
by (*auto simp*: *G-entails-def*)
qed

end
theory *Ground-Superposition-Welltypedness-Preservation*
imports *Ground-Superposition*
begin

lemma (**in** *ground-superposition-calculus*) *ground-superposition-preserves-typing*:
assumes
step: *ground-superposition D E C* **and**
wt-D: $welltyped_c$ $\mathcal{F}$ *D* **and**
wt-E: $welltyped_c$ $\mathcal{F}$ *E*
shows $welltyped_c$ $\mathcal{F}$ *C*
using *step*
proof (*cases D E C rule*: *ground-superposition.cases*)
case *hyps*: (*ground-superpositionI* L_E E' L_D D' $\mathcal{P}$ κ *t u* t')
show *?thesis*
unfolding ‹$C = add\text{-}mset\ (\mathcal{P}\ (Upair\ \kappa\langle t'\rangle_G\ u))\ (E' + D')$›
unfolding $welltyped_c$-*add-mset* $welltyped_c$-*plus*
proof (*intro conjI*)
have $\exists\tau.\ welltyped\ \mathcal{F}\ \kappa\langle t\rangle_G\ \tau \wedge welltyped\ \mathcal{F}\ u\ \tau$
proof −
have $welltyped_l$ $\mathcal{F}$ L_E
using *wt-E*
unfolding ‹$E = add\text{-}mset\ L_E\ E'$› $welltyped_c$-*add-mset*
by *argo*
hence $welltyped_a$ $\mathcal{F}$ ($Upair\ \kappa\langle t\rangle_G\ u$)
using ‹$\mathcal{P} \in \{Pos,\ Neg\}$›
unfolding ‹$L_E = \mathcal{P}\ (Upair\ \kappa\langle t\rangle_G\ u)$› $welltyped_l$-*def*
by *auto*
thus *?thesis*
unfolding $welltyped_a$-*def* **by** *simp*
qed

moreover have $\exists\tau.\ welltyped\ \mathcal{F}\ t\ \tau \wedge welltyped\ \mathcal{F}\ t'\ \tau$
proof −
have $welltyped_l$ $\mathcal{F}$ L_D
using *wt-D*
unfolding ‹$D = add\text{-}mset\ L_D\ D'$› $welltyped_c$-*add-mset*
by *argo*
hence $welltyped_a$ $\mathcal{F}$ ($Upair\ t\ t'$)
using ‹$\mathcal{P} \in \{Pos,\ Neg\}$›
unfolding ‹$L_D = t \approx t'$› $welltyped_l$-*def*
by *auto*
thus *?thesis*
unfolding $welltyped_a$-*def* **by** *simp*
qed

ultimately have $\exists\tau.$ *welltyped* $\mathcal{F}$ $\kappa\langle t'\rangle_G$ τ $\wedge$ *welltyped* $\mathcal{F}$ *u* τ
using *gctxt-apply-term-preserves-typing*[*of* $\mathcal{F}$ κ *t* - - *t*']
by *blast*
hence *welltyped*$_a$ $\mathcal{F}$ (*Upair* $\kappa\langle t'\rangle_G$ *u*)
unfolding *welltyped*$_a$*-def* **by** *simp*
thus *welltyped*$_l$ $\mathcal{F}$ ($\mathcal{P}$ (*Upair* $\kappa\langle t'\rangle_G$ *u*))
unfolding *welltyped*$_l$*-def*
using ‹$\mathcal{P} \in \{Pos, Neg\}$› **by** *auto*
next
show *welltyped*$_c$ $\mathcal{F}$ E'
using *wt-E*
unfolding ‹E = *add-mset* L_E E'› *welltyped*$_c$*-add-mset*
by *argo*
next
show *welltyped*$_c$ $\mathcal{F}$ D'
using *wt-D*
unfolding ‹D = *add-mset* L_D D'› *welltyped*$_c$*-add-mset*
by *argo*
qed
qed

lemma (**in** *ground-superposition-calculus*) *ground-eq-resolution-preserves-typing*:
assumes
step: *ground-eq-resolution* D C **and**
wt-D: *welltyped*$_c$ $\mathcal{F}$ D
shows *welltyped*$_c$ $\mathcal{F}$ C
using *step*
proof (*cases* D C *rule*: *ground-eq-resolution.cases*)
case (*ground-eq-resolutionI* L D' *t*)
thus *?thesis*
using *wt-D*
unfolding *welltyped*$_c$*-def*
by *simp*
qed

lemma (**in** *ground-superposition-calculus*) *ground-eq-factoring-preserves-typing*:
assumes
step: *ground-eq-factoring* D C **and**
wt-D: *welltyped*$_c$ $\mathcal{F}$ D
shows *welltyped*$_c$ $\mathcal{F}$ C
using *step*
proof (*cases* D C *rule*: *ground-eq-factoring.cases*)
case (*ground-eq-factoringI* L_1 L_2 D' *t* t' t'')
hence *welltyped*$_l$ $\mathcal{F}$ ($t \approx t'$) **and** *welltyped*$_l$ $\mathcal{F}$ ($t \approx t''$) **and** *welltyped*$_c$ $\mathcal{F}$ D'
unfolding *atomize-conj*
using *wt-D* *welltyped*$_c$*-add-mset* **by** *metis*

hence $\exists\tau.$ *welltyped* $\mathcal{F}$ *t* τ $\wedge$ *welltyped* $\mathcal{F}$ t' τ $\exists\tau.$ *welltyped* $\mathcal{F}$ *t* τ $\wedge$ *welltyped*

```
ℱ t'' τ
    unfolding atomize-conj welltyped_l-def welltyped_a-def by simp

  hence t-t'-same-type: ∃τ. welltyped ℱ t' τ ∧ welltyped ℱ t'' τ
    using welltyped-right-unique[THEN right-uniqueD] by metis

  show ?thesis
   unfolding ‹C = add-mset (t' !≈ t'') (add-mset (t ≈ t'') D')› welltyped_c-add-mset
  proof (intro conjI)
    show welltyped_l ℱ (t' !≈ t'')
      using t-t'-same-type
      unfolding welltyped_l-def welltyped_a-def by simp
  next
    show welltyped_l ℱ (t ≈ t'')
      using ‹welltyped_l ℱ (t ≈ t'')› .
  next
    show welltyped_c ℱ D'
      using ‹welltyped_c ℱ D'› .
  qed
qed

end
```

References

[1] M. Desharnais, B. Toth, U. Waldmann, J. Blanchette, and S. Tourret. A Modular Formalization of Superposition in Isabelle/HOL. In Y. Bertot, T. Kutsia, and M. Norrish, editors, *15th International Conference on Interactive Theorem Proving (ITP 2024)*, volume 309 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.