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*: $\bigwedge R\ x\ y.\ R^{==}\ x\ y \longleftrightarrow R\ x\ y \lor x = y$
⟨*proof*⟩

lemma *reflclp-refl*: $R^{==}\ x\ x$
⟨*proof*⟩

lemma *transpD-strict-non-strict*:
assumes *transp R*
shows $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R^{==}\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

lemma *transpD-non-strict-strict*:
assumes *transp R*
shows $\bigwedge x\ y\ z.\ R^{==}\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

lemma *mem-rtrancl-union-iff-mem-rtrancl-lhs*:
assumes $\bigwedge z.\ (x,\ z) \in A^* \Longrightarrow z \notin \mathit{Domain}\ B$
shows $(x,\ y) \in (A \cup B)^* \longleftrightarrow (x,\ y) \in A^*$
⟨*proof*⟩

lemma *mem-rtrancl-union-iff-mem-rtrancl-rhs*:
assumes
$\bigwedge z.\ (x,\ z) \in B^* \Longrightarrow z \notin \mathit{Domain}\ A$
shows $(x,\ y) \in (A \cup B)^* \longleftrightarrow (x,\ y) \in B^*$
⟨*proof*⟩

end
theory *Abstract-Rewriting-Extra*

imports
Transitive-Closure-Extra
Abstract−Rewriting.Abstract-Rewriting
begin

lemma *mem-join-union-iff-mem-join-lhs*:
assumes
$\bigwedge z.\ (x,\ z) \in A^* \Longrightarrow z \notin Domain\ B$ **and**
$\bigwedge z.\ (y,\ z) \in A^* \Longrightarrow z \notin Domain\ B$
shows $(x,\ y) \in (A \cup B)^{\downarrow} \longleftrightarrow (x,\ y) \in A^{\downarrow}$
⟨*proof*⟩

lemma *mem-join-union-iff-mem-join-rhs*:
assumes
$\bigwedge z.\ (x,\ z) \in B^* \Longrightarrow z \notin Domain\ A$ **and**
$\bigwedge z.\ (y,\ z) \in B^* \Longrightarrow z \notin Domain\ A$
shows $(x,\ y) \in (A \cup B)^{\downarrow} \longleftrightarrow (x,\ y) \in B^{\downarrow}$
⟨*proof*⟩

lemma *refl-join*: *refl* $(r^{\downarrow})$
⟨*proof*⟩

lemma *trans-join*:
assumes *strongly-norm*: *SN r* **and** *confluent*: *WCR r*
shows *trans* $(r^{\downarrow})$
⟨*proof*⟩

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

definition *compatible-with-gctxt* :: *'f gterm rel* ⇒ *bool* **where**
compatible-with-gctxt $I \longleftrightarrow (\forall\ t\ t'\ ctxt.\ (t,\ t') \in I \longrightarrow (ctxt\langle t\rangle_G,\ ctxt\langle t'\rangle_G) \in I)$

lemma *compatible-with-gctxtD*:
compatible-with-gctxt $I \Longrightarrow (t,\ t') \in I \Longrightarrow (ctxt\langle t\rangle_G,\ ctxt\langle t'\rangle_G) \in I$
⟨*proof*⟩

lemma *compatible-with-gctxt-converse*:
assumes *compatible-with-gctxt I*
shows *compatible-with-gctxt* (I^{-1})
⟨*proof*⟩

lemma *compatible-with-gctxt-symcl*:
assumes *compatible-with-gctxt I*
shows *compatible-with-gctxt* $(I^{\leftrightarrow})$
⟨*proof*⟩

lemma *compatible-with-gctxt-rtrancl*:
assumes *compatible-with-gctxt I*
shows *compatible-with-gctxt* (I^*)
⟨*proof*⟩

lemma *compatible-with-gctxt-relcomp*:
assumes *compatible-with-gctxt I1* **and** *compatible-with-gctxt I2*
shows *compatible-with-gctxt* $(I1\ O\ I2)$
⟨*proof*⟩

lemma *compatible-with-gctxt-join*:
assumes *compatible-with-gctxt I*
shows *compatible-with-gctxt* $(I^{\downarrow})$
⟨*proof*⟩

lemma *compatible-with-gctxt-conversion*:
assumes *compatible-with-gctxt I*
shows *compatible-with-gctxt* $(I^{\leftrightarrow *})$
⟨*proof*⟩

definition *rewrite-inside-gctxt* :: *'f gterm rel* ⇒ *'f gterm rel* **where**
rewrite-inside-gctxt R = $\{(ctxt\langle t1\rangle_G,\ ctxt\langle t2\rangle_G) \mid ctxt\ t1\ t2.\ (t1,\ t2) \in R\}$

lemma *mem-rewrite-inside-gctxt-if-mem-rewrite-rules*[*intro*]:
$(l,\ r) \in R \Longrightarrow (l,\ r) \in$ *rewrite-inside-gctxt R*
⟨*proof*⟩

lemma *ctxt-mem-rewrite-inside-gctxt-if-mem-rewrite-rules*[*intro*]:
$(l,\ r) \in R \Longrightarrow (ctxt\langle l\rangle_G,\ ctxt\langle r\rangle_G) \in$ *rewrite-inside-gctxt R*
⟨*proof*⟩

lemma *rewrite-inside-gctxt-mono*: $R \subseteq S \Longrightarrow$ *rewrite-inside-gctxt R* ⊆ *rewrite-inside-gctxt S*
⟨*proof*⟩

lemma *rewrite-inside-gctxt-union*:
rewrite-inside-gctxt $(R \cup S)$ = *rewrite-inside-gctxt R* ∪ *rewrite-inside-gctxt S*
⟨*proof*⟩

lemma *rewrite-inside-gctxt-insert*:
rewrite-inside-gctxt (*insert r R*) = *rewrite-inside-gctxt* {*r*} ∪ *rewrite-inside-gctxt R*
⟨*proof*⟩

lemma *converse-rewrite-steps*: $(\textit{rewrite-inside-gctxt}\ R)^{-1} = \textit{rewrite-inside-gctxt}\ (R^{-1})$
⟨*proof*⟩

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* ⟹ $ctxt\langle t2\rangle_G$ $\prec_t$ $ctxt\langle t1\rangle_G$
shows *t2* $\prec_t$ *t1*
⟨*proof*⟩

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*
⟨*proof*⟩

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* $ctxt\langle t1\rangle_G$ $ctxt\langle t2\rangle_G$
shows *P t1 t2*
⟨*proof*⟩

lemma *compatible-with-gctxt-rewrite-inside-gctxt*[*simp*]: *compatible-with-gctxt* (*rewrite-inside-gctxt E*)
⟨*proof*⟩

lemma *subset-rewrite-inside-gctxt*[*simp*]: *E* ⊆ *rewrite-inside-gctxt E*
⟨*proof*⟩

lemma *wf-converse-rewrite-inside-gctxt*:
fixes *E* :: *'f gterm rel*
assumes
wfP-R: *wfP R* **and**
R-compatible-with-gctxt: ⋀*ctxt t t'*. *R t t'* ⟹ *R* $ctxt\langle t\rangle_G$ $ctxt\langle t'\rangle_G$ **and**
equations-subset-R: ⋀*x y*. (*x*, *y*) ∈ *E* ⟹ *R y x*
shows *wf* ((*rewrite-inside-gctxt E*)$^{-1}$)
⟨*proof*⟩

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) | *ctxt l* r_1 r_2. ($ctxt\langle l\rangle_G$, r_1) ∈ *R* ∧ (*l*, r_2) ∈ *R*}

abbreviation *ground-critical-pair-theorem* :: *'f gterm rel* ⇒ *bool* **where**
ground-critical-pair-theorem (*R* :: *'f gterm rel*) ≡
WCR (*rewrite-inside-gctxt R*) ⟷ *ground-critical-pairs R* ⊆ (*rewrite-inside-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 $1 \leq count\ M\ x$
  obtains M' **where** $M = add\text{-}mset\ x\ M'$
  ⟨*proof*⟩

lemma *two-le-countE*:
  assumes $2 \leq count\ M\ x$
  obtains M' **where** $M = add\text{-}mset\ x\ (add\text{-}mset\ x\ M')$
  ⟨*proof*⟩

lemma *three-le-countE*:
  assumes $3 \leq count\ M\ x$
  obtains M' **where** $M = add\text{-}mset\ x\ (add\text{-}mset\ x\ (add\text{-}mset\ x\ M'))$
  ⟨*proof*⟩

lemma *one-step-implies-multp*$_{HO}$*-strong*:
  fixes $A\ B\ J\ K$ :: - *multiset*
  defines $J \equiv B - A$ **and** $K \equiv A - B$
  assumes $J \neq \{\#\}$ **and** $\forall k \in\# K.\ \exists x \in\# J.\ R\ k\ x$
  shows $multp_{HO}\ R\ A\ B$
  ⟨*proof*⟩

lemma *Uniq-antimono*: $Q \leq P \Longrightarrow Uniq\ Q \geq Uniq\ P$
  ⟨*proof*⟩

lemma *Uniq-antimono'*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x) \Longrightarrow Uniq\ P \Longrightarrow Uniq\ Q$
  ⟨*proof*⟩

lemma *multp-singleton-right*[*simp*]:
  assumes *transp R*
  shows $multp\ R\ M\ \{\#x\#\} \longleftrightarrow (\forall y \in\# M.\ R\ y\ x)$
⟨*proof*⟩

lemma *multp-singleton-left*[*simp*]:
  assumes *transp R*
  shows $multp\ R\ \{\#x\#\}\ M \longleftrightarrow (\{\#x\#\} \subset\# M \vee (\exists y \in\# M.\ R\ x\ y))$
⟨*proof*⟩

lemma *multp-singleton-singleton*[*simp*]: $transp\ R \Longrightarrow multp\ R\ \{\#x\#\}\ \{\#y\#\} \longleftrightarrow$

R x y
⟨*proof*⟩

lemma *multp-subset-supersetI*: *transp R* $\Longrightarrow$ *multp R A B* $\Longrightarrow$ *C* $\subseteq$# *A* $\Longrightarrow$ *B* $\subseteq$# *D* $\Longrightarrow$ *multp R C D*
⟨*proof*⟩

lemma *multp-double-doubleI*:
assumes *transp R multp R A B*
shows *multp R* (*A* + *A*) (*B* + *B*)
⟨*proof*⟩

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*⟩

lemma *multp-double-doubleD*:
assumes *transp R* **and** *asymp R* **and** *multp R* (*A* + *A*) (*B* + *B*)
shows *multp R A B*
⟨*proof*⟩

lemma *multp-double-double*:
transp R $\Longrightarrow$ *asymp R* $\Longrightarrow$ *multp R* (*A* + *A*) (*B* + *B*) $\longleftrightarrow$ *multp R A B*
⟨*proof*⟩

lemma *multp-doubleton-doubleton*[*simp*]:
transp R $\Longrightarrow$ *asymp R* $\Longrightarrow$ *multp R* {#*x*, *x*#} {#*y*, *y*#} $\longleftrightarrow$ *R x y*
⟨*proof*⟩

lemma *multp-single-doubleI*: *M* ≠ {#} $\Longrightarrow$ *multp R M* (*M* + *M*)
⟨*proof*⟩

lemma *mult1-implies-one-step-strong*:
assumes *trans r* **and** *asym r* **and** (*A*, *B*) ∈ *mult1 r*
shows *B* − *A* ≠ {#} **and** ∀ *k* ∈# *A* − *B*. ∃ *j* ∈# *B* − *A*. (*k*, *j*) ∈ *r*
⟨*proof*⟩

lemma *asymp-multp*:
assumes *asymp R* **and** *transp R*
shows *asymp* (*multp R*)
⟨*proof*⟩

lemma *multp-doubleton-singleton*: *transp R* $\Longrightarrow$ *multp R* {# *x*, *x* #} {# *y* #} $\longleftrightarrow$ *R x y*
⟨*proof*⟩

lemma *image-mset-remove1-mset*:
assumes *inj f*
shows *remove1-mset* (*f a*) (*image-mset f X*) = *image-mset f* (*remove1-mset a X*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *multp-add-mset'*:
assumes *R x y*
shows *multp R* (*add-mset x X*) (*add-mset y X*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *multp-add-same*:
assumes *asymp R transp R multp R X Y*
shows *multp R* (*add-mset x X*) (*add-mset x Y*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *ex-ordered-Upair*:
assumes *tot*: *totalp-on* (*set-uprod p*) *R*
shows ∃ *x y*. *p* = *Upair x y* ∧ $R^{==}$ *x y*
⟨*proof*⟩

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*#})
⟨*proof*⟩

lemma *set-mset-mset-uprod*[*simp*]: *set-mset* (*mset-uprod up*) = *set-uprod up*
⟨*proof*⟩

lemma *mset-uprod-Upair*[*simp*]: *mset-uprod* (*Upair x y*) = {#*x*, *y*#}
⟨*proof*⟩

lemma *map-uprod-inverse*: (⋀*x*. *f* (*g x*) = *x*) ⟹ (⋀*y*. *map-uprod f* (*map-uprod g y*) = *y*)
⟨*proof*⟩

lemma *mset-uprod-image-mset*: *mset-uprod* (*map-uprod f p*) = *image-mset f* (*mset-uprod p*)
⟨*proof*⟩

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

lemmas *UniqI* = *Uniq-I*

lemma *Uniq-prodI*:
assumes ⋀*x1 y1 x2 y2*. *P x1 y1* ⟹ *P x2 y2* ⟹ (*x1*, *y1*) = (*x2*, *y2*)
shows $\exists_{\leq 1}$(*x*, *y*). *P x y*
⟨*proof*⟩

lemma *Uniq-implies-ex1*: $\exists_{\leq 1}$*x*. *P x* ⟹ *P y* ⟹ ∃!*x*. *P x*

⟨*proof*⟩

lemma *Uniq-antimono*: $Q \leq P \Longrightarrow \mathit{Uniq}\ Q \geq \mathit{Uniq}\ P$
⟨*proof*⟩

lemma *Uniq-antimono'*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x) \Longrightarrow \mathit{Uniq}\ P \Longrightarrow \mathit{Uniq}\ Q$
⟨*proof*⟩

lemma *Collect-eq-if-Uniq*: $(\exists_{\leq 1} x.\ P\ x) \Longrightarrow \{x.\ P\ x\} = \{\} \vee (\exists\, x.\ \{x.\ P\ x\} = \{x\})$
⟨*proof*⟩

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

lemma *Ball-Ex-comm*:
$(\forall\, x \in X.\ \exists f.\ P\ (f\ x)\ x) \Longrightarrow (\exists f.\ \forall\, x \in X.\ P\ (f\ x)\ x)$
$(\exists f.\ \forall\, x \in X.\ P\ (f\ x)\ x) \Longrightarrow (\forall\, x \in X.\ \exists f.\ P\ (f\ x)\ x)$
⟨*proof*⟩

lemma *set-map-id*:
assumes $x \in \mathit{set}\ X\ f\ x \notin \mathit{set}\ X\ \ \mathit{map}\ f\ X = X$
shows *False*
⟨*proof*⟩

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

lemma *transp-on-empty*[*simp*]: *transp-on* {} *R*
⟨*proof*⟩

lemma *asymp-on-empty*[*simp*]: *asymp-on* {} *R*
⟨*proof*⟩

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*⟩

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

lemma *map-literal-inverse*:

$(\bigwedge x.\ f\ (g\ x) = x) \Longrightarrow (\bigwedge literal.\ map\text{-}literal\ f\ (map\text{-}literal\ g\ literal) = literal)$
⟨*proof*⟩

lemma *map-literal-comp*:
$map\text{-}literal\ f\ (map\text{-}literal\ g\ literal) = map\text{-}literal\ (\lambda atom.\ f\ (g\ atom))\ literal$
⟨*proof*⟩

lemma *literals-distinct* [*simp*]: $Neg \neq Pos\ Pos \neq Neg$
⟨*proof*⟩

primrec *mset-lit* :: *'a uprod literal* ⇒ *'a multiset* **where**
$mset\text{-}lit\ (Pos\ A) = mset\text{-}uprod\ A\ |$
$mset\text{-}lit\ (Neg\ A) = mset\text{-}uprod\ A + mset\text{-}uprod\ A$

lemma *mset-lit-image-mset*: $mset\text{-}lit\ (map\text{-}literal\ (map\text{-}uprod\ f)\ l) = image\text{-}mset\ f\ (mset\text{-}lit\ l)$
⟨*proof*⟩

lemma *uprod-mem-image-iff-prod-mem*[*simp*]:
assumes *sym I*
shows $(Upair\ t\ t') \in (\lambda(t_1,\ t_2).\ Upair\ t_1\ t_2)$ ‘ $I \longleftrightarrow (t,\ t') \in I$
⟨*proof*⟩

lemma *true-lit-uprod-iff-true-lit-prod*[*simp*]:
assumes *sym I*
shows
$(\lambda(t_1,\ t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models l\ Pos\ (Upair\ t\ t') \longleftrightarrow I \models l\ Pos\ (t,\ t')$
$(\lambda(t_1,\ t_2).\ Upair\ t_1\ t_2)$ ‘ $I \models l\ Neg\ (Upair\ t\ t') \longleftrightarrow I \models l\ Neg\ (t,\ t')$
⟨*proof*⟩

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

lemma *gterm-is-fun*: *is-Fun* (*term-of-gterm t*)
⟨*proof*⟩

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

lemma *le-size-gctxt*: $size\ t \leq size\ (C\langle t\rangle_G)$
⟨*proof*⟩

lemma *lt-size-gctxt*: $ctxt \neq \Box_G \Longrightarrow size\ t < size\ ctxt\langle t\rangle_G$
⟨*proof*⟩

lemma *gctxt-ident-iff-eq-GHole*[*simp*]: $ctxt\langle t\rangle_G = t \longleftrightarrow ctxt = \square_G$
⟨*proof*⟩

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: $\bigwedge C.$ *sel C* ⊆# *C* **and**
select-negative-lits: $\bigwedge C\ L.$ *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* $R^{==}$
⟨*proof*⟩

lemma *order-reflclp-if-transp-and-asymp*:

assumes *transp R* **and** *asymp R*
shows *class.order* $R^{==}$ *R*
⟨*proof*⟩

locale *term-ordering-lifting* =
fixes
less-trm :: *'t* ⇒ *'t* ⇒ *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* ⇒ *'t uprod literal* ⇒ *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* ⇒ *'t uprod clause* ⇒ *bool* (**infix** $\prec_c$ *50*) **where**
less-cls ≡ *multp* ($\prec_l$)

sublocale *term-order*: *order* $(\prec_t)^{==}$ ($\prec_t$)
⟨*proof*⟩

sublocale *literal-order*: *order* $(\prec_l)^{==}$ ($\prec_l$)
⟨*proof*⟩

sublocale *clause-order*: *order* $(\prec_c)^{==}$ ($\prec_c$)
⟨*proof*⟩

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* ⇒ *'f gterm* ⇒ *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*: $t \preceq_t ctxt\langle t\rangle_G$
⟨*proof*⟩

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* $(\prec_l)$
⟨*proof*⟩

lemma *wfP-less-cls*[*simp*]: *wfp* $(\prec_c)$
⟨*proof*⟩

sublocale *term-order*: *linorder lesseq-trm less-trm*
⟨*proof*⟩

sublocale *literal-order*: *linorder lesseq-lit less-lit*
⟨*proof*⟩

sublocale *clause-order*: *linorder lesseq-cls less-cls*
⟨*proof*⟩

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*⟩

lemma *less-trm-compatible-with-gctxt-iff*: $ctxt\langle t\rangle_G \prec_t ctxt\langle t'\rangle_G \longleftrightarrow t \prec_t t'$
⟨*proof*⟩

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 $ctxt\langle t\rangle_G \prec_t ctxt'\langle t'\rangle_G$

$\langle proof \rangle$

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 $ctxt\langle t\rangle_G \prec_t ctxt'\langle t'\rangle_G$
$\langle proof \rangle$

end

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

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

lemma *welltyped-right-unique*: $right\text{-}unique\ (welltyped\ \mathcal{F})$
$\langle proof \rangle$

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

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

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*:
$welltyped_c\ \mathcal{F}\ (add\text{-}mset\ L\ C) \longleftrightarrow welltyped_l\ \mathcal{F}\ L \wedge welltyped_c\ \mathcal{F}\ C$
$\langle proof \rangle$

lemma $welltyped_c$*-plus*:
$welltyped_c\ \mathcal{F}\ (C + D) \longleftrightarrow welltyped_c\ \mathcal{F}\ C \wedge welltyped_c\ \mathcal{F}\ D$
$\langle proof \rangle$

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 $welltyped\ \mathcal{F}\ \kappa\langle t'\rangle_G\ \tau_1$

⟨*proof*⟩

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: ⋀(*R* :: *'f gterm rel*). *ground-critical-pair-theorem*
R
begin

1.1 Ground Rules

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

inductive *ground-eq-resolution* ::
$'f$ *gatom clause* $\Rightarrow$ $'f$ *gatom clause* $\Rightarrow$ *bool* **where**
ground-eq-resolutionI:
$D = \textit{add-mset}\ L\ D' \Longrightarrow$
$L = Neg\ (Upair\ t\ t) \Longrightarrow$
$\textit{select}\ D = \{\#\} \wedge \textit{is-maximal-lit}\ L\ D \vee \textit{is-maximal-lit}\ L\ (\textit{select}\ D) \Longrightarrow$
$C = D' \Longrightarrow$
ground-eq-resolution $D\ C$

inductive *ground-eq-factoring* ::
$'f$ *gatom clause* $\Rightarrow$ $'f$ *gatom clause* $\Rightarrow$ *bool* **where**
ground-eq-factoringI:
$D = \textit{add-mset}\ L_1\ (\textit{add-mset}\ L_2\ D') \Longrightarrow$
$L_1 = t \approx t' \Longrightarrow$
$L_2 = t \approx t'' \Longrightarrow$
$\textit{select}\ D = \{\#\} \Longrightarrow$
$\textit{is-maximal-lit}\ L_1\ D \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$C = \textit{add-mset}\ (Neg\ (Upair\ t'\ t''))\ (\textit{add-mset}\ (t \approx t'')\ D') \Longrightarrow$
ground-eq-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:
$P_1 = \textit{add-mset}\ L_1\ {P_1}' \Longrightarrow$

$P_2 = \textit{add-mset}\ L_2\ P_2{}' \Longrightarrow$
$P_2 \prec_c P_1 \Longrightarrow$
$\mathcal{P} \in \{Pos,\ Neg\} \Longrightarrow$
$L_1 = \mathcal{P}\ (Upair\ s\langle t\rangle_G\ s') \Longrightarrow$
$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
$(\mathcal{P} = Pos \longrightarrow \textit{select}\ P_1 = \{\#\} \wedge \textit{is-strictly-maximal-lit}\ L_1\ P_1) \Longrightarrow$
$(\mathcal{P} = Neg \longrightarrow (\textit{select}\ P_1 = \{\#\} \wedge \textit{is-maximal-lit}\ L_1\ P_1 \vee \textit{is-maximal-lit}\ L_1$ $(\textit{select}\ P_1))) \Longrightarrow$
$\textit{select}\ P_2 = \{\#\} \Longrightarrow$
$\textit{is-strictly-maximal-lit}\ L_2\ P_2 \Longrightarrow$
$C = \textit{add-mset}\ (\mathcal{P}\ (Upair\ s\langle t'\rangle_G\ s'))\ (P_1{}' + P_2{}') \Longrightarrow$
$\textit{ground-superposition}'\ P_2\ P_1\ C$

lemma *ground-superposition′ = ground-superposition*
⟨*proof*⟩

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

lemma *ground-superposition-if-ground-pos-superposition*:
assumes *step*: *ground-pos-superposition* $P_2\ P_1\ C$
shows *ground-superposition* $P_2\ P_1\ C$
⟨*proof*⟩

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

$L_2 = t \approx t' \Longrightarrow$
$s' \prec_t s\langle t\rangle_G \Longrightarrow$
$t' \prec_t t \Longrightarrow$
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
⟨*proof*⟩

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*⟩

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 $N_2 \longleftrightarrow$ (∀ (I :: *'f gterm rel*). *refl* $I \longrightarrow$ *trans* $I \longrightarrow$ *sym* $I \longrightarrow$
compatible-with-gctxt $I \longrightarrow$ *upair* ' I ⊫*s* $N_1 \longrightarrow$ *upair* ' I ⊫*s* N_2)

lemma *ground-superposition-smaller-conclusion*:
assumes
step: *ground-superposition P1 P2 C*
shows $C \prec_c$ *P2*
⟨*proof*⟩

lemma *ground-eq-resolution-smaller-conclusion*:
assumes *step*: *ground-eq-resolution* P C
shows $C \prec_c P$
⟨*proof*⟩

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

end

sublocale *ground-superposition-calculus* $\subseteq$ *consequence-relation* **where**
Bot = *G-Bot* **and**
entails = *G-entails*
⟨*proof*⟩

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

1.3 Redundancy Criterion

sublocale *ground-superposition-calculus* $\subseteq$ *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*⟩

1.4 Mode Construction

context *ground-superposition-calculus* **begin**

function *epsilon* :: - $\Rightarrow$ *'f gatom clause* $\Rightarrow$ *'f gterm rel* **where**
epsilon N C = $\{(s, t)|$ *s t C'*.
$C \in N \wedge$
C = *add-mset* (*Pos* (*Upair s t*)) $C' \wedge$
select C = $\{\#\} \wedge$
is-strictly-maximal-lit (*Pos* (*Upair s t*)) $C \wedge$
$t \prec_t s \wedge$
(*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* $R_C)^{\downarrow} \models C \wedge$
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (s, t) $R_C))^{\downarrow} \models C' \wedge$
$s \in$ *NF* (*rewrite-inside-gctxt* R_C))}
⟨*proof*⟩

termination *epsilon*
⟨*proof*⟩

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

lemma *epsilon-filter-le-conv*: *epsilon* $\{D \in N.\ D \preceq_c C\}$ *C* = *epsilon N C*
⟨*proof*⟩

end

lemma (**in** *ground-ordering*) *Uniq-striclty-maximal-lit-in-ground-cls*:
$\exists_{\leq 1} L.$ *is-strictly-maximal-lit* L C
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *epsilon-eq-empty-or-singleton*:
epsilon N $C = \{\} \vee (\exists s\ t.$ *epsilon* N $C = \{(s, t)\})$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *card-epsilon-le-one*:
card (*epsilon* N C) ≤ 1
⟨*proof*⟩

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
⟨*proof*⟩

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))$^{\downarrow} \models C$ **and**
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (l, r) (*rewrite-sys* N C)))$^{\downarrow} \models C'$ **and**
$l \in NF$ (*rewrite-inside-gctxt* (*rewrite-sys* N C))
⟨*proof*⟩

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'* $\{D \in N.\ D \prec_c C\}$))$^{\downarrow} \models C \wedge$
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* (*insert* (l, r) (*rewrite-sys'* $\{D \in N.\ D \prec_c C\}$)))$^{\downarrow}$ $\models C' \wedge$
$l \in NF$ (*rewrite-inside-gctxt* (*rewrite-sys'* $\{D \in N.\ D \prec_c C\}$)))
(**is** *?LHS* $\longleftrightarrow$ *?RHS*)
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *rhs-lt-lhs-if-mem-rewrite-sys*:

assumes $(t1, t2) \in$ *rewrite-sys N C*
shows $t2 \prec_t t1$
⟨*proof*⟩

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*⟩

lemma (**in** *ground-superposition-calculus*) *rhs-lesseq-trm-lhs-if-mem-rtrancl-rewrite-inside-gctxt-rewrite-sys*:
assumes *rule-in*: $(t1, t2) \in$ (*rewrite-inside-gctxt* (*rewrite-sys N C*))*
shows $t2 \preceq_t t1$
⟨*proof*⟩

lemma *singleton-eq-CollectD*: $\{x\} = \{y.\ P\ y\} \Longrightarrow P\ x$
⟨*proof*⟩

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

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma (**in** *ground-superposition-calculus*) *termination-rewrite-sys*: *wf* ((*rewrite-sys N C*)$^{-1}$)
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *termination-Union-rewrite-sys*:
wf $((\bigcup D \in N.$ *rewrite-sys N D*$)^{-1})$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *no-crit-pairs*:
$\{(t1, t2) \in$ *ground-critical-pairs* $(\bigcup$ (*epsilon N2* ‘ N)). $t1 \neq t2\} = \{\}$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *WCR-Union-rewrite-sys*:
WCR (*rewrite-inside-gctxt* $(\bigcup D \in N.$ *epsilon N2 D*))
⟨*proof*⟩

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*⟩

lemma (**in** *ground-ordering*) *less-trm-const-lhs-if-mem-rewrite-inside-gctxt*:
fixes *t t1 t2 r*
assumes
rule-in: $(t1, t2) \in$ *rewrite-inside-gctxt* r **and**
ball-lt-lhs: $\bigwedge t1\ t2.\ (t1, t2) \in r \Longrightarrow t \prec_t t1$
shows $t \prec_t t1$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *split-Union-epsilon*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.$ *epsilon* $N\ C) =$
rewrite-sys $N\ D \cup$ *epsilon* $N\ D \cup (\bigcup C \in \{C \in N.\ D \prec_c C\}.$ *epsilon* $N\ C)$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *split-Union-epsilon'*:
assumes *D-in*: $D \in N$
shows $(\bigcup C \in N.$ *epsilon* $N\ C) =$ *rewrite-sys* $N\ D \cup (\bigcup C \in \{C \in N.\ D \preceq_c C\}.$
epsilon $N\ C)$
⟨*proof*⟩

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-sys* $N\ C =$ *rewrite-sys* $N\ D \cup (\bigcup C' \in \{C' \in N.\ D \preceq_c C' \wedge C'$
$\prec_c C\}.$ *epsilon* $N\ C')$
⟨*proof*⟩

lemma (**in** *ground-ordering*) *mem-join-union-iff-mem-join-lhs'*:
assumes
ball-R_1*-rhs-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_1 \Longrightarrow t2 \prec_t t1$ **and**
ball-R_2*-lt-lhs*: $\bigwedge t1\ t2.\ (t1, t2) \in R_2 \Longrightarrow s \prec_t t1 \wedge t \prec_t t1$
shows $(s, t) \in (R_1 \cup R_2)^\downarrow \longleftrightarrow (s, t) \in R_1^\downarrow$
⟨*proof*⟩

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 $(s, t) \in (R_1 \cup R_2)^{\downarrow} \longleftrightarrow (s, t) \in R_2{}^{\downarrow}$
⟨*proof*⟩

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*⟩

lemma (**in** *ground-superposition-calculus*) *lift-entailment-to-Union*:
fixes *N D*
defines $R_D \equiv \mathit{rewrite\text{-}sys}\ N\ D$
assumes
D-in: $D \in N$ **and**
R_D-entails-D: $\mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ R_D)^{\downarrow} \models D$
shows
$\mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\bigcup D \in N.\ \mathit{epsilon}\ N\ D))^{\downarrow} \models D$ **and**
$\bigwedge C.\ C \in N \Longrightarrow D \prec_c C \Longrightarrow \mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\mathit{rewrite\text{-}sys}\ N\ C))^{\downarrow} \models D$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*)
assumes *productive*: $\mathit{epsilon}\ N\ C = \{(l, r)\}$
shows
true-cls-if-productive-epsilon:
$\mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\bigcup D \in N.\ \mathit{epsilon}\ N\ D))^{\downarrow} \models C$
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow \mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\mathit{rewrite\text{-}sys}\ N\ D))^{\downarrow} \models C$ **and**
false-cls-if-productive-epsilon:
$\neg\ \mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\bigcup D \in N.\ \mathit{epsilon}\ N\ D))^{\downarrow} \models C - \{\#\mathit{Pos}\ (\mathit{Upair}\ l\ r)\#\}$
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow \neg\ \mathit{upair}\ \text{`}\ (\mathit{rewrite\text{-}inside\text{-}gctxt}\ (\mathit{rewrite\text{-}sys}\ N\ D))^{\downarrow} \models C - \{\#\mathit{Pos}\ (\mathit{Upair}\ l\ r)\#\}$
⟨*proof*⟩

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** $(w, z) \in r^*$
⟨*proof*⟩

lemma *ex-step-if-joinable*:
assumes $\mathit{asymp}\ R\ (x, z) \in r^*$ **and** $(y, z) \in r^*$
shows
$R^{==}\ z\ y \Longrightarrow R\ y\ x \Longrightarrow \exists\, w.\ (x, w) \in r \wedge (w, z) \in r^*$

$R^{==}\ z\ x \Longrightarrow R\ x\ y \Longrightarrow \exists\, w.\ (y,\ w) \in r \wedge (w,\ z) \in r^*$
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *trans-join-rewrite-inside-gctxt-rewrite-sys*:
trans ((*rewrite-inside-gctxt* (*rewrite-sys* N C))$^\downarrow$)
⟨*proof*⟩

lemma (**in** *ground-ordering*) *true-cls-insert-and-not-true-clsE*:
assumes
upair ‘ (*rewrite-inside-gctxt* (*insert* r R))$^\downarrow$ $\Vdash$ C **and**
$\neg$ *upair* ‘ (*rewrite-inside-gctxt* R)$^\downarrow$ $\Vdash$ C
obtains t t' **where**
Pos (*Upair* t t') $\in\#$ 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*⟩

lemma (**in** *ground-superposition-calculus*) *model-preconstruction*:
fixes
N :: *'f gatom clause set* **and**
C :: *'f gatom clause*
defines
entails $\equiv \lambda E\ C.$ *upair* ‘ (*rewrite-inside-gctxt* E)$^\downarrow$ $\Vdash$ C
assumes *saturated* N **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
shows
epsilon N C = {} $\longleftrightarrow$ *entails* (*rewrite-sys* N C) C
$\bigwedge D.\ D \in N \Longrightarrow C \prec_c D \Longrightarrow$ *entails* (*rewrite-sys* N D) C
⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *model-construction*:
fixes
N :: *'f gatom clause set* **and**
C :: *'f gatom clause*
defines
entails $\equiv \lambda E\ C.$ *upair* ‘ (*rewrite-inside-gctxt* E)$^\downarrow$ $\Vdash$ C
assumes *saturated* N **and** $\{\#\} \notin N$ **and** *C-in*: $C \in N$
shows *entails* ($\bigcup D \in N.$ *epsilon* N D) C
⟨*proof*⟩

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 $\{\#\} \in N$
⟨*proof*⟩

sublocale *ground-superposition-calculus* $\subseteq$ *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*⟩

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*]: $\bigwedge b$. *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*}
⟨*proof*⟩

lemma *fset-finite-set* [*simp*]: *fset* (*finite-set b*) = *set b*
⟨*proof*⟩

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: $\bigwedge a\ \sigma\ \tau.\ (\bigwedge x.\ x \in (vars\ a) \Longrightarrow \sigma\ x = \tau\ x) \Longrightarrow a \cdot \sigma = a \cdot \tau$
begin

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

definition *vars-set* :: *'expression set* ⇒ *'variable set* **where**
vars-set expressions ≡ $\bigcup$ *expression* ∈ *expressions*. *vars expression*

lemma *subst-reduntant-upd* [*simp*]:
assumes *var* ∉ *vars a*
shows *a* · σ(*var* := *update*) = *a* · σ
⟨*proof*⟩

lemma *subst-reduntant-if* [*simp*]:
assumes $vars\ a \subseteq vars'$
shows $a \cdot (\lambda var.\ if\ var \in vars'\ then\ \sigma\ var\ else\ \sigma'\ var) = a \cdot \sigma$
⟨*proof*⟩

lemma *subst-reduntant-if'* [*simp*]:
assumes $vars\ a \cap vars' = \{\}$
shows $a \cdot (\lambda var.\ if\ var \in vars'\ then\ \sigma'\ var\ else\ \sigma\ var) = a \cdot \sigma$
⟨*proof*⟩

lemma *subst-cannot-unground*:
assumes $\neg is\text{-}ground\ (a \cdot \sigma)$
shows $\neg is\text{-}ground\ a$
⟨*proof*⟩

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: $\bigwedge a.\ is\text{-}ground\ a \longleftrightarrow (\forall \sigma.\ subst\ a\ \sigma = a)$ **and**
exists-non-ident-subst:
$\bigwedge a\ s.\ finite\ s \Longrightarrow \neg is\text{-}ground\ a \Longrightarrow \exists \sigma.\ subst\ a\ \sigma \neq a \wedge subst\ a\ \sigma \notin s$

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: $\{f.\ is\text{-}ground\ f\} = range\ from\text{-}ground$ **and**
from-ground-inverse [*simp*]: $\bigwedge g.\ to\text{-}ground\ (from\text{-}ground\ g) = g$
begin

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

lemma *to-ground-from-ground-id*: $to\text{-}ground \circ from\text{-}ground = id$
⟨*proof*⟩

lemma *surj-to-ground*: *surj to-ground*
⟨*proof*⟩

lemma *inj-from-ground*: *inj-on from-ground* $domain_G$
⟨*proof*⟩

lemma *inj-on-to-ground*: *inj-on to-ground* (*from-ground* ‘ $domain_G$)
⟨*proof*⟩

lemma *bij-betw-to-ground*: *bij-betw to-ground* (*from-ground* ‘ $domain_G$) $domain_G$
⟨*proof*⟩

lemma *bij-betw-from-ground*: *bij-betw from-ground* $domain_G$ (*from-ground* ‘ $domain_G$)
⟨*proof*⟩

lemma *ground-is-ground* [*simp*, *intro*]: *is-ground* (*from-ground g*)
⟨*proof*⟩

lemma *is-ground-iff-range-from-ground*: *is-ground f* ⟷ *f* ∈ *range from-ground*
⟨*proof*⟩

lemma *to-ground-inverse* [*simp*]:
assumes *is-ground f*
shows *from-ground* (*to-ground f*) = *f*
⟨*proof*⟩

corollary *obtain-grounding*:
assumes *is-ground f*
obtains *g* **where** *from-ground g* = *f*
⟨*proof*⟩

end

locale *base-variable-substitution* = *variable-substitution*
where *subst* = *subst*
for *subst* :: ′*expression* ⇒ (′*variable* ⇒ ′*expression*) ⇒ ′*expression* (**infixl** · *70*)
+
assumes
is-grounding-iff-vars-grounded:
⋀*exp*. *is-ground* (*exp* · γ) ⟷ (∀ *x* ∈ *vars exp*. *is-ground* (γ *x*)) **and**
ground-exists: ∃ *exp*. *is-ground exp*
begin

lemma *obtain-ground-subst*:
obtains γ
where *is-ground-subst* γ
⟨*proof*⟩

lemma *ground-subst-extension*:
assumes *is-ground* (*exp* · γ)
obtains γ'
where *exp* · γ = *exp* · γ' **and** *is-ground-subst* γ'
⟨*proof*⟩

lemma *ground-subst-upd* [*simp*]:
assumes *is-ground update is-ground* (*exp* · γ)
shows *is-ground* (*exp* · γ(*var* := *update*))
⟨*proof*⟩

lemma *variable-grounding*:
assumes *is-ground* (t · γ) $x \in$ *vars* t
shows *is-ground* (γ x)
⟨*proof*⟩

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
⟨*proof*⟩

lemma *ground-subst-extension*:
assumes *is-ground* (*exp* · γ)
obtains γ'
where *exp* · γ = *exp* · γ' **and** *is-ground-subst* γ'
⟨*proof*⟩

lemma *ground-subst-extension*$'$:
assumes *is-ground* (*exp* · γ)
obtains γ'
where *exp* · γ = *exp* · γ' **and** *base.is-ground-subst* γ'
⟨*proof*⟩

lemma *ground-subst-upd* [*simp*]:
assumes *base.is-ground update is-ground* (*exp* · γ)

shows *is-ground* $(exp \cdot \gamma(var := update))$
⟨*proof*⟩

lemma *ground-exists*: $\exists\, exp.\ is\text{-}ground\ exp$
⟨*proof*⟩

lemma *variable-grounding*:
assumes $is\text{-}ground\ (t \cdot \gamma)\ x \in vars\ t$
shows $base.is\text{-}ground\ (\gamma\ x)$
⟨*proof*⟩

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: $\bigwedge d\ f\ g.\ (\bigwedge c.\ c \in to\text{-}set\ d \Longrightarrow f\ c = g\ c) \Longrightarrow map\ f\ d = map\ g\ d$
and
to-set-map: $\bigwedge d\ f.\ to\text{-}set\ (map\ f\ d) = f\ `\ to\text{-}set\ d$ **and**
exists-expression: $\bigwedge c.\ \exists\, d.\ c \in to\text{-}set\ d$
begin

definition *vars* :: *'expression* ⇒ *'variable set* **where**
$vars\ d \equiv \bigcup\ (sub\text{-}vars\ `\ to\text{-}set\ d)$

definition *subst* :: *'expression* ⇒ (*'variable* ⇒ *'base-expression*) ⇒ *'expression*
where
$subst\ d\ \sigma \equiv map\ (\lambda c.\ sub\text{-}subst\ c\ \sigma)\ d$

lemma *map-id-cong*:
assumes $\bigwedge c.\ c \in to\text{-}set\ d \Longrightarrow f\ c = c$
shows $map\ f\ d = d$
⟨*proof*⟩

lemma *to-set-map-not-ident*:
assumes $c \in to\text{-}set\ d\ f\ c \notin to\text{-}set\ d$
shows $map\ f\ d \neq d$

⟨*proof*⟩

lemma *subst-in-to-set-subst*:
assumes $c \in$ *to-set d*
shows *sub-subst c σ* $\in$ *to-set* (*subst d σ*)
⟨*proof*⟩

sublocale *variable-substitution* **where** *subst* = *subst* **and** *vars* = *vars*
⟨*proof*⟩

lemma *ground-subst-iff-sub-ground-subst* [*simp*]:
is-ground-subst γ ⟷ *sub.is-ground-subst γ*
⟨*proof*⟩

lemma *to-set-is-ground* [*intro*]:
assumes *sub* $\in$ *to-set expr is-ground expr*
shows *sub.is-ground sub*
⟨*proof*⟩

lemma *to-set-is-ground-subst*:
assumes *sub* $\in$ *to-set expr* *is-ground* (*subst expr γ*)
shows *sub.is-ground* (*sub-subst sub γ*)
⟨*proof*⟩

lemma *subst-empty*:
assumes *to-set expr′* = {}
shows *subst expr σ* = *expr′* ⟷ *expr* = *expr′*
⟨*proof*⟩

lemma *empty-is-ground*:
assumes *to-set expr* = {}
shows *is-ground expr*
⟨*proof*⟩

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) $\longleftrightarrow$ ($\forall$ *x* $\in$ *vars exp*. *base.is-ground* (γ *x*))
⟨*proof*⟩

lemma *ground-subst-iff-base-ground-subst* [*simp*]:
$\bigwedge\gamma$. *is-ground-subst* γ $\longleftrightarrow$ *base.is-ground-subst* γ
⟨*proof*⟩

lemma *obtain-ground-subst*:
obtains γ
where *is-ground-subst* γ
⟨*proof*⟩

lemma *ground-subst-extension*:
assumes *is-ground* (*subst exp* γ)
obtains γ'
where *subst exp* γ = *subst exp* γ' **and** *is-ground-subst* γ'
⟨*proof*⟩

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'
⟨*proof*⟩

lemma *ground-subst-upd* [*simp*]:
assumes *base.is-ground update is-ground* (*subst exp* γ)
shows *is-ground* (*subst exp* (γ(*var* := *update*)))
⟨*proof*⟩

lemma *ground-exists*: $\exists$ *exp*. *is-ground exp*
⟨*proof*⟩

lemma *variable-grounding*:
assumes *is-ground* (*subst t* γ) *x* $\in$ *vars t*
shows *base.is-ground* (γ *x*)
⟨*proof*⟩

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* $\Rightarrow$ *'c fset* **where**
to-fset $\equiv$ *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*
⟨*proof*⟩

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*⟩

lemma *to-set-from-ground*: *to-set* (*from-ground expr*) = *sub-from-ground* ‘ (*to-set-ground expr*)
⟨*proof*⟩

lemma *sub-in-ground-is-ground*:

assumes *sub* ∈ *to-set* (*from-ground expr*)
shows *sub.is-ground sub*
⟨*proof*⟩

lemma *ground-sub-in-ground*:
sub ∈ *to-set-ground expr* ⟷ *sub-from-ground sub* ∈ *to-set* (*from-ground expr*)
⟨*proof*⟩

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))
⟨*proof*⟩

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*⟩

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** · *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*]: $(\forall t \in set\text{-}uprod\ (Upair\ t_1\ t_2).\ P\ t) \longleftrightarrow P\ t_1 \wedge P\ t_2$
⟨*proof*⟩

lemma *infinite-terms* [*clause-intro*]: *infinite* (*UNIV* :: (*'f*, *'v*) *term set*)
⟨*proof*⟩

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

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* ⇒ *'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*⟨*proof*⟩

global-interpretation *context*: *vars-def* **where**
vars-def = *vars-ctxt*⟨*proof*⟩

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

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

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* = $(\cdot t)$
⟨*proof*⟩

lemma *term-context-ground-iff-term-is-ground* [*clause-simp*]:
Term-Context.ground t = *term.is-ground t*
⟨*proof*⟩

global-interpretation
term: *grounding* **where**
vars = *term.vars* :: (*'f*, *'v*) *term* ⇒ *'v set* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
subst = $(\cdot t)$ **and** *to-ground* = *term.to-ground* **and** *from-ground* = *term.from-ground*
⟨*proof*⟩

global-interpretation *context*: *all-subst-ident-iff-ground* **where**
is-ground = λκ. *context.vars* κ = {} **and** *subst* = $(\cdot t_c)$
⟨*proof*⟩

global-interpretation *context*: *based-variable-substitution* **where**

subst = (·t_c) **and** *vars* = *context.vars* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
base-vars = *term.vars* **and** *base-subst* = (·*t*)
⟨*proof*⟩

global-interpretation *context*: *finite-variables*
where *vars* = *context.vars* :: ('*f*, '*v*) *context* ⇒ '*v set*
⟨*proof*⟩

global-interpretation *context*: *grounding* **where**
vars = *context.vars* :: ('*f*, '*v*) *context* ⇒ '*v set* **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
subst = (·t_c) **and** *from-ground* = *context.from-ground* **and** *to-ground* = *context.to-ground*
⟨*proof*⟩

lemma *ground-ctxt-iff-context-is-ground* [*clause-simp*]:
ground-ctxt context ⟷ *context.is-ground context*
⟨*proof*⟩

5 Lifting

lemma *exists-uprod*: ∃ *a*. *t* ∈ *set-uprod a*
⟨*proof*⟩

lemma *exists-literal*: ∃ *l*. *a* ∈ *set-literal l*
⟨*proof*⟩

lemma *exists-mset*: ∃ *c*. *l* ∈ *set-mset c*
⟨*proof*⟩

lemma *finite-set-literal*: ⋀*l*. *finite* (*set-literal l*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mset-lit-set-literal* [*clause-simp*]:
term ∈# *mset-lit literal* ⟷ *term* ∈ ⋃(*set-uprod* ‘ *set-literal literal*)
⟨*proof*⟩

lemma *vars-atom* [*clause-simp*]:
atom.vars (*Upair* $term_1$ $term_2$) = *term.vars* $term_1$ ∪ *term.vars* $term_2$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *subst-atom* [*clause-simp*]:
$Upair\ term_1\ term_2 \cdot a\ \sigma = Upair\ (term_1 \cdot t\ \sigma)\ (term_2 \cdot t\ \sigma)$
⟨*proof*⟩

lemma *subst-literal* [*clause-simp*]:
$Pos\ atom \cdot l\ \sigma = Pos\ (atom \cdot a\ \sigma)$
$Neg\ atom \cdot l\ \sigma = Neg\ (atom \cdot a\ \sigma)$
$atm\text{-}of\ (literal \cdot l\ \sigma) = atm\text{-}of\ literal \cdot a\ \sigma$
⟨*proof*⟩

lemma *vars-clause-add-mset* [*clause-simp*]:
$clause.vars\ (add\text{-}mset\ literal\ clause) = literal.vars\ literal \cup clause.vars\ clause$
⟨*proof*⟩

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

lemma *clause-submset-vars-clause-subset* [*clause-intro*]:
$clause_1 \subseteq\#\ clause_2 \Longrightarrow clause.vars\ clause_1 \subseteq clause.vars\ clause_2$
⟨*proof*⟩

lemma *subst-clause-add-mset* [*clause-simp*]:
$add\text{-}mset\ literal\ clause \cdot \sigma = add\text{-}mset\ (literal \cdot l\ \sigma)\ (clause \cdot \sigma)$
⟨*proof*⟩

lemma *subst-clause-plus* [*clause-simp*]:
$(clause_1 + clause_2) \cdot \sigma = clause_1 \cdot \sigma + clause_2 \cdot \sigma$
⟨*proof*⟩

lemma *clause-to-ground-plus* [*simp*]:
$clause.to\text{-}ground\ (clause_1 + clause_2) = clause.to\text{-}ground\ clause_1 + clause.to\text{-}ground\ clause_2$
⟨*proof*⟩

lemma *clause-from-ground-plus* [*simp*]:
$clause.from\text{-}ground\ (clause_{G1} + clause_{G2}) = clause.from\text{-}ground\ clause_{G1} + clause.from\text{-}ground\ clause_{G2}$
⟨*proof*⟩

lemma *subst-clause-remove1-mset* [*clause-simp*]:
assumes $literal \in\#\ clause$
shows $remove1\text{-}mset\ literal\ clause \cdot \sigma = remove1\text{-}mset\ (literal \cdot l\ \sigma)\ (clause \cdot \sigma)$
⟨*proof*⟩

lemma *sub-ground-clause* [*clause-intro*]:
assumes *clause'* $\subseteq\#$ *clause clause.is-ground clause*
shows *clause.is-ground clause'*
⟨*proof*⟩

lemma *clause-from-ground-empty-mset* [*clause-simp*]: *clause.from-ground* {#} = {#}
⟨*proof*⟩

lemma *clause-to-ground-empty-mset* [*clause-simp*]: *clause.to-ground* {#} = {#}
⟨*proof*⟩

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*⟩
⟨*proof*⟩

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*$_G$⟩
⟨*proof*⟩

lemma *ground-term-with-context3*:
(*context.from-ground context*$_G$)⟨*term.from-ground term*$_G$⟩ = *term.from-ground context*$_G$⟨*term*$_G$⟩$_G$
⟨*proof*⟩

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'*
⟨*proof*⟩

lemma *context-is-ground-context-compose2*:
assumes *context.is-ground context context.is-ground context'*
shows *context.is-ground* (*context* $\circ_c$ *context'*)
⟨*proof*⟩

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

lemma *ground-context-subst*:

assumes
$context.is\text{-}ground\ context_G$
$context_G = (context \cdot t_c\ \sigma) \circ_c context'$
shows
$context_G = context \circ_c context' \cdot t_c\ \sigma$
⟨*proof*⟩

lemma *clause-from-ground-add-mset* [*clause-simp*]:
$clause.from\text{-}ground\ (add\text{-}mset\ literal_G\ clause_G) =$
$add\text{-}mset\ (literal.from\text{-}ground\ literal_G)\ (clause.from\text{-}ground\ clause_G)$
⟨*proof*⟩

lemma *remove1-mset-literal-from-ground*:
$remove1\text{-}mset\ (literal.from\text{-}ground\ literal_G)\ (clause.from\text{-}ground\ clause_G)$
$= clause.from\text{-}ground\ (remove1\text{-}mset\ literal_G\ clause_G)$
⟨*proof*⟩

lemma *term-with-context-is-ground* [*clause-simp*]:
$term.is\text{-}ground\ context\langle term\rangle \longleftrightarrow context.is\text{-}ground\ context \wedge term.is\text{-}ground\ term$
⟨*proof*⟩

lemma *mset-literal-from-ground*:
$mset\text{-}lit\ (literal.from\text{-}ground\ l) = image\text{-}mset\ term.from\text{-}ground\ (mset\text{-}lit\ l)$
⟨*proof*⟩

lemma *clause-is-ground-add-mset* [*clause-simp*]:
$clause.is\text{-}ground\ (add\text{-}mset\ literal\ clause) \longleftrightarrow$
$literal.is\text{-}ground\ literal \wedge clause.is\text{-}ground\ clause$
⟨*proof*⟩

lemma *clause-to-ground-add-mset*:
assumes $clause.from\text{-}ground\ clause = add\text{-}mset\ literal\ clause'$
shows $clause = add\text{-}mset\ (literal.to\text{-}ground\ literal)\ (clause.to\text{-}ground\ clause')$
⟨*proof*⟩

lemma *mset-mset-lit-subst* [*clause-simp*]:
$\{\#\ term \cdot t\ \sigma.\ term \in\#\ mset\text{-}lit\ literal\ \#\} = mset\text{-}lit\ (literal \cdot l\ \sigma)$
⟨*proof*⟩

lemma *term-in-literal-subst* [*clause-intro*]:
assumes $term \in\#\ mset\text{-}lit\ literal$
shows $term \cdot t\ \sigma \in\#\ mset\text{-}lit\ (literal \cdot l\ \sigma)$
⟨*proof*⟩

lemma *ground-term-in-ground-literal*:
assumes $literal.is\text{-}ground\ literal\ term \in\#\ mset\text{-}lit\ literal$

shows *term.is-ground term*
⟨*proof*⟩

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)
⟨*proof*⟩

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*
⟨*proof*⟩

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* $term_{G2}$)
⟨*proof*⟩

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* $atom_G$)
⟨*proof*⟩

lemma *context-from-ground-hole* [*clause-simp*]:
context.from-ground $context_G$ = □ ⟷ $context_G$ = $□_G$
⟨*proof*⟩

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* $literal_G$)
⟨*proof*⟩

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* $term_1$ $term_2$)
⟨*proof*⟩

lemma *ground-terms-in-ground-atom2* [*clause-simp*]:
atom.is-ground (*Upair* $term_1$ $term_2$) ⟷ *term.is-ground* $term_1$ ∧ *term.is-ground* $term_2$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *atom-is-ground-in-ground-literal* [*intro*]:
literal.is-ground literal ⟷ *atom.is-ground* (*atm-of literal*)
⟨*proof*⟩

lemma *obtain-from-atom-subst* [*clause-intro*]:
assumes *Upair* $term_1{}'$ $term_2{}'$ = *atom* ·*a* σ
obtains $term_1$ $term_2$
where *atom* = *Upair* $term_1$ $term_2$ $term_1{}' = term_1 \cdot t\ \sigma$ $term_2{}' = term_2 \cdot t\ \sigma$
⟨*proof*⟩

lemma *obtain-from-pos-literal-subst* [*clause-intro*]:
assumes *literal* ·*l* σ = $term_1{}' \approx term_2{}'$
obtains $term_1$ $term_2$
where *literal* = $term_1 \approx term_2$ $term_1{}' = term_1 \cdot t\ \sigma$ $term_2{}' = term_2 \cdot t\ \sigma$
⟨*proof*⟩

lemma *obtain-from-neg-literal-subst*:
assumes *literal* ·*l* σ = $term_1{}'$!≈ $term_2{}'$
obtains $term_1$ $term_2$
where *literal* = $term_1$!≈ $term_2$ $term_1 \cdot t\ \sigma = term_1{}'$ $term_2 \cdot t\ \sigma = term_2{}'$
⟨*proof*⟩

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

lemma *subst-cannot-add-var*:
assumes *is-Var* (*term* ·*t* σ)
shows *is-Var term*
⟨*proof*⟩

lemma *var-in-term*:
assumes *var* ∈ *term.vars term*
obtains *context* **where** *term* = *context*⟨*Var var*⟩
⟨*proof*⟩

lemma *var-in-non-ground-term*:
assumes ¬ *term.is-ground term*
obtains *context var* **where** *term* = *context*⟨*var*⟩ *is-Var var*
⟨*proof*⟩

lemma *non-ground-arg*:

assumes ¬ *term.is-ground* (*Fun f terms*)
obtains *term*
where *term* ∈ *set terms* ¬ *term.is-ground term*
⟨*proof*⟩

lemma *non-ground-arg′*:
assumes ¬ *term.is-ground* (*Fun f terms*)
obtains *ts1 var ts2*
where *terms* = *ts1* @ [*var*] @ *ts2* ¬ *term.is-ground var*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *interpretation-context-congruence′*:
assumes
(*t*, *t′*) ∈ *I*
(*ctxt*⟨*t*⟩$_G$, *t″*) ∉ *I*
shows
(*ctxt*⟨*t′*⟩$_G$, *t″*) ∉ *I*
⟨*proof*⟩

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* ·*t* γ)
begin

lemma *interpretation-term-congruence*:
assumes
term-grounding: *term.is-ground* (*term* ·*t* γ) **and**
var-update: (*term.to-ground* (γ *var*), *term.to-ground update*) $\in I$ **and**
updated-term: (*term.to-ground* (*term* ·*t* γ(*var* := *update*)), *term*$'$) $\in I$
shows
(*term.to-ground* (*term* ·*t* γ), *term*$'$) $\in I$
⟨*proof*⟩

lemma *interpretation-term-congruence*$'$:
assumes
term-grounding: *term.is-ground* (*term* ·*t* γ) **and**
var-update: (*term.to-ground* (γ *var*), *term.to-ground update*) $\in I$ **and**
updated-term: (*term.to-ground* (*term* ·*t* γ(*var* := *update*)), *term*$'$) $\notin I$
shows
(*term.to-ground* (*term* ·*t* γ), *term*$'$) $\notin I$
⟨*proof*⟩

lemma *interpretation-atom-congruence*:
assumes
term.is-ground (*term*$_1$ ·*t* γ)
term.is-ground (*term*$_2$ ·*t* γ)
(*term.to-ground* (γ *var*), *term.to-ground update*) $\in I$
(*term.to-ground* (*term*$_1$ ·*t* γ(*var* := *update*)), *term.to-ground* (*term*$_2$ ·*t* γ(*var* := *update*))) $\in I$
shows
(*term.to-ground* (*term*$_1$ ·*t* γ), *term.to-ground* (*term*$_2$ ·*t* γ)) $\in I$
⟨*proof*⟩

lemma *interpretation-atom-congruence*$'$:
assumes
term.is-ground (*term*$_1$ ·*t* γ)
term.is-ground (*term*$_2$ ·*t* γ)
(*term.to-ground* (γ *var*), *term.to-ground update*) $\in I$
(*term.to-ground* (*term*$_1$ ·*t* γ(*var* := *update*)), *term.to-ground* (*term*$_2$ ·*t* γ(*var* := *update*))) $\notin I$
shows
(*term.to-ground* (*term*$_1$ ·*t* γ), *term.to-ground* (*term*$_2$ ·*t* γ)) $\notin I$
⟨*proof*⟩

lemma *interpretation-literal-congruence*:
assumes
literal.is-ground (*literal* ·*l* γ)
upair ‘ *I* $\models$*l* *term.to-ground* (*Var var* ·*t* γ) $\approx$ *term.to-ground update*
upair ‘ *I* $\models$*l* *literal.to-ground* (*literal* ·*l* γ(*var* := *update*))

shows
upair ‘ I $\Vdash$*l literal.to-ground* (*literal* ·*l* γ)
⟨*proof*⟩

lemma *interpretation-clause-congruence*:
assumes
clause.is-ground (*clause* · γ)
upair ‘ I $\Vdash$*l term.to-ground* (*Var var* ·*t* γ) ≈ *term.to-ground update*
upair ‘ I $\Vdash$ *clause.to-ground* (*clause* · γ(*var* := *update*))
shows
upair ‘ I $\Vdash$ *clause.to-ground* (*clause* · γ)
⟨*proof*⟩

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*⟩

lemma *renaming-vars-atom*: *Var ‘ atom.vars* (*atom* ·*a* ϱ) = ϱ ‘ *atom.vars atom*
⟨*proof*⟩

lemma *renaming-vars-literal*: *Var ‘ literal.vars* (*literal* ·*l* ϱ) = ϱ ‘ *literal.vars literal*
⟨*proof*⟩

lemma *renaming-vars-clause*: *Var ‘ clause.vars* (*clause* · ϱ) = ϱ ‘ *clause.vars clause*
⟨*proof*⟩

lemma *surj-the-inv*: *surj* (λ*x*. *the-inv* ϱ (*Var x*))
⟨*proof*⟩

end

lemma *needed*: *surj g* ⟹ *infinite* {*x*. *f x* = *ty*} ⟹ *infinite* {*x*. *f* (*g x*) = *ty*}
⟨*proof*⟩

lemma *obtain-ground-fun*:
assumes *term.is-ground t*
obtains *f ts* **where** *t* = *Fun f ts*
⟨*proof*⟩

lemma *vars-term-subst*: *term.vars* (*t* ·*t* σ) ⊆ *term.vars t* ∪ *range-vars* σ

⟨*proof*⟩

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'*
⟨*proof*⟩

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'*
⟨*proof*⟩

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'*
⟨*proof*⟩

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'*
⟨*proof*⟩

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'*
⟨*proof*⟩

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*⟩

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*⟩

lemma *obtain-inj-on*:

assumes *finite domain infinite image-subset*
obtains *f*
where
inj-on (f :: $'a \Rightarrow 'b$) *domain*
f ‘ *domain* $\subseteq$ *image-subset*
⟨*proof*⟩

corollary *obtain-inj-on'*:
assumes *finite domain infinite* (*UNIV* :: $'b$ *set*)
obtains *f*
where *inj-on* (f :: $'a \Rightarrow 'b$) *domain*
⟨*proof*⟩

corollary *obtain-inj*:
assumes *finite* (*UNIV* :: $'a$ *set*) *infinite* (*UNIV* :: $'b$ *set*)
obtains *f*
where *inj* (f :: $'a \Rightarrow 'b$)
⟨*proof*⟩

corollary *obtain-inj'*:
assumes *finite* (*UNIV* :: $'a$ *set*) *infinite image-subset*
obtains *f*
where *inj* (f :: $'a \Rightarrow 'b$) f ‘ *domain* $\subseteq$ *image-subset*
⟨*proof*⟩

lemma *obtain-inj-endo*:
assumes *finite domain infinite image-subset*
obtains f :: $'a \Rightarrow 'a$
where *inj f f* ‘ *domain* $\subseteq$ *image-subset*
⟨*proof*⟩

abbreviation *surj-on* **where**
surj-on domain f $\equiv$ ($\forall\, y.\ \exists\, x \in$ *domain*. $y = f\ x$)

lemma *surj-on-alternative*: *surj-on domain f* $\longleftrightarrow$ f ‘ *domain* $=$ *UNIV*
⟨*proof*⟩

lemma *obtain-surj-on-nat*:
assumes *infinite domain*
obtains f :: $'a \Rightarrow$ *nat* **where** *surj-on domain f*
⟨*proof*⟩

lemma *obtain-surj-on*:
assumes *infinite domain*
obtains f :: $'a \Rightarrow 'b$ :: *countable* **where** *surj-on domain f*
⟨*proof*⟩

lemma *partitions*:
assumes *infinite* (*UNIV* :: $'x$ *set*)

obtains A B **where**
$|A| =o\ |B|$
$|A| =o\ |UNIV :: 'x\ set|$
$A \cap B = \{\}$
$A \cup B = (UNIV :: 'x\ set)$
$\langle proof \rangle$

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* $\mathcal{F}\ \mathcal{V}\ (Var\ x)\ \tau$
| *Fun*: $\mathcal{F}\ f = (\tau s, \tau) \Longrightarrow$ *has-type* $\mathcal{F}\ \mathcal{V}\ (Fun\ f\ ts)\ \tau$

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* $\mathcal{F}\ \mathcal{V}\ (Var\ x)\ \tau$
| *Fun*: $\mathcal{F}\ f = (\tau s, \tau) \Longrightarrow$ *list-all2* (*welltyped* $\mathcal{F}\ \mathcal{V}$) $ts\ \tau s \Longrightarrow$ *welltyped* $\mathcal{F}\ \mathcal{V}\ (Fun\ f\ ts)\ \tau$

lemma *has-type-right-unique*: *right-unique* (*has-type* $\mathcal{F}\ \mathcal{V}$)
$\langle proof \rangle$

lemma *welltyped-right-unique*: *right-unique* (*welltyped* $\mathcal{F}\ \mathcal{V}$)
$\langle proof \rangle$

definition $has\text{-}type_a$ **where**
$has\text{-}type_a\ \mathcal{F}\ \mathcal{V}\ A \longleftrightarrow (\exists \tau.\ \forall t \in set\text{-}uprod\ A.\ has\text{-}type\ \mathcal{F}\ \mathcal{V}\ t\ \tau)$

definition $welltyped_a$ **where**
[*clause-simp*]: $welltyped_a\ \mathcal{F}\ \mathcal{V}\ A \longleftrightarrow (\exists \tau.\ \forall t \in set\text{-}uprod\ A.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau)$

definition $has\text{-}type_l$ **where**
$has\text{-}type_l\ \mathcal{F}\ \mathcal{V}\ L \longleftrightarrow has\text{-}type_a\ \mathcal{F}\ \mathcal{V}\ (atm\text{-}of\ L)$

definition $welltyped_l$ **where**
[*clause-simp*]: $welltyped_l\ \mathcal{F}\ \mathcal{V}\ L \longleftrightarrow welltyped_a\ \mathcal{F}\ \mathcal{V}\ (atm\text{-}of\ L)$

definition $has\text{-}type_c$ **where**
$has\text{-}type_c\ \mathcal{F}\ \mathcal{V}\ C \longleftrightarrow (\forall L \in\#\ C.\ has\text{-}type_l\ \mathcal{F}\ \mathcal{V}\ L)$

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*
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \wedge$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ C
⟨*proof*⟩

lemma *welltyped*$_c$*-add-mset* [*clause-simp*]:
welltyped$_c$ $\mathcal{F}$ $\mathcal{V}$ (*add-mset* L C) $\longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \wedge$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ C
⟨*proof*⟩

lemma *has-type*$_c$*-plus* [*clause-simp*]:
has-type$_c$ $\mathcal{F}$ $\mathcal{V}$ $(C + D) \longleftrightarrow$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ $C \wedge$ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ D
⟨*proof*⟩

lemma *welltyped*$_c$*-plus* [*clause-simp*]:
welltyped$_c$ $\mathcal{F}$ $\mathcal{V}$ (C + D) $\longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ C $\wedge$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ D
⟨*proof*⟩

lemma *has-type*$_\sigma$*-has-type*:
assumes *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ *has-type* $\mathcal{F}$ $\mathcal{V}$ t τ
shows *has-type* $\mathcal{F}$ $\mathcal{V}$ (t $\cdot t$ σ) τ
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-welltyped*:
assumes *welltyped*$_\sigma$: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ (t $\cdot t$ σ) τ $\longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ
⟨*proof*⟩

lemma *has-type*$_\sigma$*-has-type*$_a$:
assumes *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ *has-type*$_a$ $\mathcal{F}$ $\mathcal{V}$ a
shows *has-type*$_a$ $\mathcal{F}$ $\mathcal{V}$ (a $\cdot a$ σ)
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-welltyped*$_a$:
assumes *welltyped*$_\sigma$: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ (a $\cdot a$ σ) $\longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ a
⟨*proof*⟩

lemma *has-type*$_\sigma$*-has-type*$_l$:
assumes *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ *has-type*$_l$ $\mathcal{F}$ $\mathcal{V}$ l
shows *has-type*$_l$ $\mathcal{F}$ $\mathcal{V}$ (l $\cdot l$ σ)
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-welltyped*$_l$:
assumes *welltyped*$_\sigma$: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ (l $\cdot l$ σ) $\longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ l
⟨*proof*⟩

lemma *has-type*$_\sigma$*-has-type*$_c$:
assumes *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ c
shows *has-type*$_c$ $\mathcal{F}$ $\mathcal{V}$ (c $\cdot$ σ)
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-on-welltyped*:
assumes *wt*: *welltyped*$_\sigma$*-on* (*term.vars* t) $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ (t $\cdot t$ σ) τ $\longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-on-welltyped*$_a$:
assumes *wt*: *welltyped*$_\sigma$*-on* (*atom.vars* A) $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ (A $\cdot a$ σ) $\longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ A
⟨*proof*⟩

lemma *welltyped*$_l$*-iff-welltyped*$_a$: *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $L \longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ (*atm-of* L)
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-on-welltyped*$_l$:
assumes *wt*: *welltyped*$_\sigma$*-on* (*literal.vars* L) $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ ($L \cdot l\ \sigma$) $\longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ L
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-on-welltyped*$_c$:
assumes *wt*: *welltyped*$_\sigma$*-on* (*clause.vars* C) $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ ($C \cdot \sigma$) $\longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ C
⟨*proof*⟩

lemma *welltyped*$_\sigma$*-welltyped*$_c$:
assumes *welltyped*$_\sigma$: *welltyped*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ σ
shows *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ ($c \cdot \sigma$) $\longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ c
⟨*proof*⟩

lemma *has-type*$_\kappa$:
assumes
κ*-type*: *has-type* $\mathcal{F}$ $\mathcal{V}$ $\kappa\langle t\rangle$ τ_1 **and**
t-type: *has-type* $\mathcal{F}$ $\mathcal{V}$ t τ_2 **and**
t'*-type*: *has-type* $\mathcal{F}$ $\mathcal{V}$ t' τ_2
shows
has-type $\mathcal{F}$ $\mathcal{V}$ $\kappa\langle t'\rangle$ τ_1
⟨*proof*⟩

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 τ'
⟨*proof*⟩

lemma *welltyped*$_\kappa$*'*:
assumes *welltyped* $\mathcal{F}$ $\mathcal{V}$ $\kappa\langle t\rangle$ τ
shows $\exists \tau'$. *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ'
⟨*proof*⟩

lemma *welltyped*$_\kappa$ [*clause-intro*]:
assumes
κ*-type*: *welltyped* $\mathcal{F}$ $\mathcal{V}$ $\kappa\langle t\rangle$ τ_1 **and**
t-type: *welltyped* $\mathcal{F}$ $\mathcal{V}$ t τ_2 **and**
t'*-type*: *welltyped* $\mathcal{F}$ $\mathcal{V}$ t' τ_2
shows
welltyped $\mathcal{F}$ $\mathcal{V}$ $\kappa\langle t'\rangle$ τ_1
⟨*proof*⟩

lemma *has-type*$_\sigma$*-Var*: *has-type*$_\sigma$ $\mathcal{F}$ $\mathcal{V}$ *Var*
⟨*proof*⟩

lemma *welltyped-add-literal*:
  assumes $welltyped_c\ \mathcal{F}\ \mathcal{V}\ P'$ *welltyped* $\mathcal{F}\ \mathcal{V}\ s_1\ \tau$ *welltyped* $\mathcal{F}\ \mathcal{V}\ s_2\ \tau$
  shows $welltyped_c\ \mathcal{F}\ \mathcal{V}\ (add\text{-}mset\ (s_1\ !\approx\ s_2)\ P')$
  ⟨*proof*⟩

lemma *welltyped-*$\mathcal{V}$:
  assumes
    $\forall x \in term.vars\ t.\ \mathcal{V}\ x = \mathcal{V}'\ x$
    welltyped $\mathcal{F}\ \mathcal{V}\ t\ \tau$
  shows
    welltyped $\mathcal{F}\ \mathcal{V}'\ t\ \tau$
  ⟨*proof*⟩

lemma *welltyped-subst-*$\mathcal{V}$:
  assumes
    $\forall x \in X.\ \mathcal{V}\ x = \mathcal{V}'\ x$
    $\forall x \in X.\ term.is\text{-}ground\ (\gamma\ x)$
  shows
    $welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}\ \gamma \longleftrightarrow welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}'\ \gamma$
  ⟨*proof*⟩

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
    $welltyped_a\ \mathcal{F}\ \mathcal{V}'\ a$
  ⟨*proof*⟩

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
    $welltyped_l\ \mathcal{F}\ \mathcal{V}'\ l$
  ⟨*proof*⟩

lemma $welltyped_c$-$\mathcal{V}$:
  assumes
    $\forall x \in clause.vars\ c.\ \mathcal{V}\ x = \mathcal{V}'\ x$
    $welltyped_c\ \mathcal{F}\ \mathcal{V}\ c$
  shows
    $welltyped_c\ \mathcal{F}\ \mathcal{V}'\ c$
  ⟨*proof*⟩

lemma *welltyped-renaming*′:
  assumes
    $term\text{-}subst.is\text{-}renaming\ \varrho$

$welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ ϱ
welltyped typeof-fun $(\lambda x.\ \mathcal{V}\ (\mathit{the\text{-}inv}\ \mathit{Var}\ (\varrho\ x)))\ t\ \tau$
shows *welltyped typeof-fun* $\mathcal{V}\ (t \cdot t\ \varrho)\ \tau$
⟨*proof*⟩

lemma $welltyped_a$*-renaming′*:
assumes
term-subst.is-renaming ϱ
$welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ ϱ
$welltyped_a$ *typeof-fun* $(\lambda x.\ \mathcal{V}\ (\mathit{the\text{-}inv}\ \mathit{Var}\ (\varrho\ x)))\ a$
shows $welltyped_a$ *typeof-fun* $\mathcal{V}\ (a \cdot a\ \varrho)$
⟨*proof*⟩

lemma $welltyped_l$*-renaming′*:
assumes
term-subst.is-renaming ϱ
$welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ ϱ
$welltyped_l$ *typeof-fun* $(\lambda x.\ \mathcal{V}\ (\mathit{the\text{-}inv}\ \mathit{Var}\ (\varrho\ x)))\ l$
shows $welltyped_l$ *typeof-fun* $\mathcal{V}\ (l \cdot l\ \varrho)$
⟨*proof*⟩

lemma $welltyped_c$*-renaming′*:
assumes
term-subst.is-renaming ϱ
$welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ ϱ
$welltyped_c$ *typeof-fun* $(\lambda x.\ \mathcal{V}\ (\mathit{the\text{-}inv}\ \mathit{Var}\ (\varrho\ x)))\ c$
shows $welltyped_c$ *typeof-fun* $\mathcal{V}\ (c \cdot \varrho)$
⟨*proof*⟩

definition *range-vars′* :: $('f,\ 'v)\ \mathit{subst} \Rightarrow\ 'v\ \mathit{set}$ **where**
range-vars′ $\sigma = \bigcup(\mathit{term.vars}\ `\ \mathit{range}\ \sigma)$

lemma *vars-term-range-vars′*:
assumes $x \in \mathit{term.vars}\ (t \cdot t\ \sigma)$
shows $x \in$ *range-vars′* σ
⟨*proof*⟩

context
fixes $\varrho\ \mathcal{V}\ \mathcal{V}'$
assumes
renaming: *term-subst.is-renaming* ϱ **and**
range-vars: $\forall x \in$ *range-vars′* $\varrho.\ \mathcal{V}\ (\mathit{the\text{-}inv}\ \varrho\ (\mathit{Var}\ x)) = \mathcal{V}'\ x$
begin

lemma *welltyped-renaming*: *welltyped* $\mathcal{F}\ \mathcal{V}\ t\ \tau \longleftrightarrow$ *welltyped* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
⟨*proof*⟩

lemma *has-type-renaming*: *has-type* $\mathcal{F}\ \mathcal{V}\ t\ \tau \longleftrightarrow$ *has-type* $\mathcal{F}\ \mathcal{V}'\ (t \cdot t\ \varrho)\ \tau$
⟨*proof*⟩

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*⟩

lemma *welltyped*$_a$*-renaming*: *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}$ $a \longleftrightarrow$ *welltyped*$_a$ $\mathcal{F}$ $\mathcal{V}'$ ($a \cdot a\ \varrho$)
  ⟨*proof*⟩

lemma *welltyped*$_l$*-renaming*: *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}$ $l \longleftrightarrow$ *welltyped*$_l$ $\mathcal{F}$ $\mathcal{V}'$ ($l \cdot l\ \varrho$)
  ⟨*proof*⟩

lemma *welltyped*$_c$*-renaming*: *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}$ $c \longleftrightarrow$ *welltyped*$_c$ $\mathcal{F}$ $\mathcal{V}'$ ($c \cdot \varrho$)
  ⟨*proof*⟩

end

context
  fixes ϱ
  assumes *renaming*: *term-subst.is-renaming* ϱ
begin

lemma *welltyped-renaming-weaker*:
  assumes $\forall x \in$ *term.vars* ($t \cdot t\ \varrho$). $\mathcal{V}$ (*the-inv* ϱ (*Var* x)) = $\mathcal{V}'$ x
  shows *welltyped* $\mathcal{F}$ $\mathcal{V}$ t $\tau \longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}'$ ($t \cdot t\ \varrho$) τ
⟨*proof*⟩

lemma *welltyped*$_a$*-renaming-weaker*:
  assumes$\forall x \in$ *atom.vars* ($a \cdot a\ \varrho$). $\mathcal{V}$ (*the-inv* ϱ (*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*⟩

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*$_l$ $\mathcal{F}$ $\mathcal{V}'$ ($l \cdot l\ \varrho$)
  ⟨*proof*⟩

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*$_c$ $\mathcal{F}$ $\mathcal{V}'$ ($c \cdot \varrho$)
  ⟨*proof*⟩

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* $\mathcal{F}$ $\mathcal{V}'$ ($t \cdot t\ \varrho$) τ
  ⟨*proof*⟩

lemma *welltyped*$_\sigma$*-renaming-ground-subst-weaker*:

assumes
welltyped$_\sigma$ $\mathcal{F}$ $\mathcal{V}'$ γ
welltyped$_\sigma$*-on* X $\mathcal{F}$ $\mathcal{V}$ ϱ
term-subst.is-ground-subst γ
$\forall x \in \bigcup(\mathit{term.vars}$ ‘ 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*⟩

end

lemma
infinite-even-nat: *infinite* { n :: *nat* . *even* n } **and**
infinite-odd-nat: *infinite* { n :: *nat* . *odd* n }
⟨*proof*⟩

lemma *obtain-infinite-partition*:
obtains X Y :: *'a* :: {*countable*, *infinite*} *set*
where
$X \cap Y = \{\}$ $X \cup Y = \mathit{UNIV}$ **and**
infinite X **and**
infinite Y
⟨*proof*⟩

lemma $(\bigcup n'.\{ n.\ g\ n = n' \}) = \mathit{UNIV}$
⟨*proof*⟩

lemma *inv-enumerate*:
assumes *infinite* N
shows (λx. *inv* (*enumerate* N) x) ‘ N = *UNIV*
⟨*proof*⟩

instance *nat* :: *infinite*
⟨*proof*⟩

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}
⟨*proof*⟩

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 $f\ f' :: nat \Rightarrow nat$ **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*⟩

lemma *obtain-inj′′-on′*:
fixes $\mathcal{V}_1\ \mathcal{V}_2 :: 'a :: infinite \Rightarrow 'ty$
assumes *finite X finite Y* $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\ x = ty\}$ $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty\}$
obtains $f\ f' :: 'a \Rightarrow 'a$ **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*⟩

lemma *obtain-inj′′-on*:
fixes $\mathcal{V}_1\ \mathcal{V}_2 :: 'a :: \{countable,\ infinite\} \Rightarrow 'ty$
assumes *finite X finite Y* $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_1\ x = ty\}$ $\bigwedge ty.$ *infinite* $\{x.\ \mathcal{V}_2\ x = ty\}$
obtains $f\ f' :: 'a \Rightarrow 'a$ **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*⟩

lemma *obtain-inj′*:
obtains $f :: 'a :: infinite \Rightarrow 'a$ **where**
inj f
$|range\ f| =o\ |UNIV - range\ f|$
⟨*proof*⟩

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*⟩

lemma *obtain-injs*:
obtains $f\, f' :: 'a :: \{countable,\ infinite\} \Rightarrow 'a$ **where**
$inj\ f\ inj\ f'$
$range\ f \cap range\ f' = \{\}$
$range\ f \cup range\ f' = UNIV$
⟨*proof*⟩

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 $\varrho_1\ \varrho_2 :: ('f,\ 'v :: infinite)\ subst$ **where**
$term\text{-}subst.is\text{-}renaming\ \varrho_1$
$term\text{-}subst.is\text{-}renaming\ \varrho_2$
$\varrho_1\ `\ X \cap \varrho_2\ `\ Y = \{\}$
$welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}_1\ \varrho_1$
$welltyped_\sigma\text{-}on\ Y\ \mathcal{F}\ \mathcal{V}_2\ \varrho_2$
⟨*proof*⟩

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 $\varrho_1\ \varrho_2 :: ('f,\ 'v :: \{countable,\ infinite\})\ subst$ **where**
$term\text{-}subst.is\text{-}renaming\ \varrho_1$
$term\text{-}subst.is\text{-}renaming\ \varrho_2$
$\varrho_1\ `\ X \cap \varrho_2\ `\ Y = \{\}$
$welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}_1\ \varrho_1$
$welltyped_\sigma\text{-}on\ Y\ \mathcal{F}\ \mathcal{V}_2\ \varrho_2$
⟨*proof*⟩

lemma $welltyped_\sigma$*-subst-upd*:
assumes $welltyped\ \mathcal{F}\ \mathcal{V}\ (Var\ var)\ \tau\ welltyped\ \mathcal{F}\ \mathcal{V}\ update\ \tau\ \ welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ \gamma$
shows $welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ (\gamma(var := update))$
⟨*proof*⟩

lemma $welltyped_\sigma$*-on-subst-upd*:
assumes $welltyped\ \mathcal{F}\ \mathcal{V}\ (Var\ var)\ \tau\ welltyped\ \mathcal{F}\ \mathcal{V}\ update\ \tau\ \ welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}\ \gamma$
shows $welltyped_\sigma\text{-}on\ X\ \mathcal{F}\ \mathcal{V}\ (\gamma(var := update))$
⟨*proof*⟩

lemma *welltyped-is-ground*:
assumes $term.is\text{-}ground\ t\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau$
shows $welltyped\ \mathcal{F}\ \mathcal{V}'\ t\ \tau$
⟨*proof*⟩

lemma *term-subst-is-imgu-is-mgu*: $term\text{-}subst.is\text{-}imgu\ \mu\ \{\{s,\ t\}\} = is\text{-}imgu\ \mu\ \{(s,\ t)\}$

⟨*proof*⟩

lemma *the-mgu-term-subst-is-imgu*:
fixes σ :: ('*f*, '*v*) *subst*
assumes $s \cdot t\ \sigma = t \cdot t\ \sigma$
shows *term-subst.is-imgu* (*the-mgu s t*) $\{\{s, t\}\}$
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *welltyped-unify*:

assumes
$unify\ es\ bs = Some\ unifier$
$\forall (t, t') \in set\ es.\ \exists \tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}\ t'\ \tau$
$welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ (subst\text{-}of\ bs)$
shows $welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ (subst\text{-}of\ unifier)$
⟨*proof*⟩

lemma *welltyped-unify'*:
assumes
unify: $unify\ [(t, t')]\ [] = Some\ unifier$ **and**
τ: $\exists \tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}\ t'\ \tau$
shows $welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ (subst\text{-}of\ unifier)$
⟨*proof*⟩

lemma *welltyped-the-mgu*:
assumes
the-mgu: $the\text{-}mgu\ t\ t' = \mu$ **and**
τ: $\exists \tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ t\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}\ t'\ \tau$
shows
$welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ \mu$
⟨*proof*⟩

abbreviation *welltyped-imgu* **where**
$welltyped\text{-}imgu\ \mathcal{F}\ \mathcal{V}\ term\ term'\ \mu \equiv$
$\forall \tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ term\ \tau \longrightarrow welltyped\ \mathcal{F}\ \mathcal{V}\ term'\ \tau \longrightarrow welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ \mu$

lemma *welltyped-imgu-exists*:
fixes $\upsilon :: ('f, 'v)\ subst$
assumes *unified*: $term \cdot t\ \upsilon = term' \cdot t\ \upsilon$
obtains $\mu :: ('f, 'v)\ subst$
where
$\upsilon = \mu \odot \upsilon$
$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{term, term'\}\}$
$welltyped\text{-}imgu\ \mathcal{F}\ \mathcal{V}\ term\ term'\ \mu$
⟨*proof*⟩

abbreviation *welltyped-imgu'* **where**
$welltyped\text{-}imgu'\ \mathcal{F}\ \mathcal{V}\ term\ term'\ \mu \equiv$
$\exists \tau.\ welltyped\ \mathcal{F}\ \mathcal{V}\ term\ \tau \wedge welltyped\ \mathcal{F}\ \mathcal{V}\ term'\ \tau \wedge welltyped_\sigma\ \mathcal{F}\ \mathcal{V}\ \mu$

lemma *welltyped-imgu'-exists*:
fixes $\upsilon :: ('f, 'v)\ subst$
assumes *unified*: $term \cdot t\ \upsilon = term' \cdot t\ \upsilon$ **and** $welltyped\ \mathcal{F}\ \mathcal{V}\ term\ \tau\ welltyped\ \mathcal{F}\ \mathcal{V}\ term'\ \tau$
obtains $\mu :: ('f, 'v)\ subst$
where
$\upsilon = \mu \odot \upsilon$
$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{term, term'\}\}$
$welltyped\text{-}imgu'\ \mathcal{F}\ \mathcal{V}\ term\ term'\ \mu$

⟨*proof*⟩

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* $\times$ $('v, 'ty)$ *var-types*

type-synonym *'f ground-select* $=$ *'f ground-atom clause* $\Rightarrow$ *'f ground-atom clause*
type-synonym $('f, 'v)$ *select* $=$ $('f, 'v)$ *atom clause* $\Rightarrow$ $('f, 'v)$ *atom clause*

definition *is-select-grounding* :: $('f, 'v)$ *select* $\Rightarrow$ *'f ground-select* $\Rightarrow$ *bool* **where**
$\bigwedge select\ select_G.$
$is\text{-}select\text{-}grounding\ select\ select_G = (\forall\, clause_G.\ \exists\, clause\ \gamma.$
$clause.is\text{-}ground\ (clause \cdot \gamma)\ \wedge$
$clause_G = clause.to\text{-}ground\ (clause \cdot \gamma) \wedge$
$select_G\ clause_G = clause.to\text{-}ground\ ((select\ clause) \cdot \gamma))$

lemma *infinite-lists-per-length*: *infinite* $\{l :: ('a :: infinite)\ list.\ length\ (tl\ l) = y\}$
⟨*proof*⟩

lemma *infinite-prods'*: $\{p :: 'a \times 'a\ .\ fst\ p = y\} = \{y\} \times UNIV$
⟨*proof*⟩

lemma *infinite-prods*: *infinite* $\{p :: (('a :: infinite) \times 'a).\ fst\ p = y\}$
⟨*proof*⟩

lemma *nat-version'*: $\exists f :: nat \Rightarrow nat.\ \forall\, y :: nat.\ infinite\ \{x.\ f\ x = y\}$
⟨*proof*⟩

lemma *not-nat-version'*: $\exists f :: ('a :: infinite) \Rightarrow 'a.\ \forall\, y.\ infinite\ \{x.\ f\ x = y\}$
⟨*proof*⟩

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*⟩

lemma *nat-version*: $\exists f :: nat \Rightarrow nat.\ \forall\, y :: nat.\ infinite\ \{x.\ f\ x = y\}$
⟨*proof*⟩

definition *all-types* **where**

all-types $\mathcal{V} \equiv \forall\, ty.\ infinite\ \{x.\ \mathcal{V}\ x = ty\}$

lemma *all-types-nat*: $\exists\,\mathcal{V} :: nat \Rightarrow nat.\ all\text{-}types\ \mathcal{V}$
⟨*proof*⟩

lemma *all-types*: $\exists\,\mathcal{V} :: ('v :: \{infinite,\ countable\} \Rightarrow 'ty :: countable).\ all\text{-}types\ \mathcal{V}$
⟨*proof*⟩

lemma *all-types'*:
assumes $|UNIV :: 'ty\ set| \leq o\ |UNIV :: ('v :: infinite)\ set|$
shows $\exists\,\mathcal{V} :: ('v :: infinite \Rightarrow 'ty).\ all\text{-}types\ \mathcal{V}$
⟨*proof*⟩

definition *clause-groundings* :: $('f,\ 'ty)\ fun\text{-}types \Rightarrow ('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow 'f\ ground\text{-}atom\ clause\ set$ **where**
$clause\text{-}groundings\ \mathcal{F}\ clause = \{\ clause.to\text{-}ground\ (fst\ clause \cdot \gamma) \mid \gamma.$
$term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma\ \wedge$
$welltyped_c\ \mathcal{F}\ (snd\ clause)\ (fst\ clause)\ \wedge$
$welltyped_\sigma\text{-}on\ (clause.vars\ (fst\ clause))\ \ \mathcal{F}\ (snd\ clause)\ \gamma\ \wedge$
$all\text{-}types\ (snd\ clause)$
$\}$

abbreviation *select-subst-stability-on* **where**
$\bigwedge select\ select_G.\ select\text{-}subst\text{-}stability\text{-}on\ \mathcal{F}\ select\ select_G\ premises \equiv$
$\forall\, premise_G \in \bigcup\ (clause\text{-}groundings\ \mathcal{F}\ `\ premises).\ \exists\,(premise,\ \mathcal{V}) \in premises.$
$\exists\,\gamma.$
$premise \cdot \gamma = clause.from\text{-}ground\ premise_G\ \wedge$
$select_G\ (clause.to\text{-}ground\ (premise \cdot \gamma)) = clause.to\text{-}ground\ ((select\ premise)$
$\cdot\ \gamma)\ \wedge$
$welltyped_c\ \mathcal{F}\ \mathcal{V}\ premise \wedge welltyped_\sigma\text{-}on\ (clause.vars\ premise)\ \mathcal{F}\ \mathcal{V}\ \gamma\ \wedge$
$term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma\ \ \wedge$
$all\text{-}types\ \mathcal{V}$

lemma *obtain-subst-stable-on-select-grounding*:
fixes $select :: ('f,\ 'v)\ select$
obtains $select_G$ **where**
$select\text{-}subst\text{-}stability\text{-}on\ \mathcal{F}\ select\ select_G\ premises$
$is\text{-}select\text{-}grounding\ select\ select_G$
⟨*proof*⟩

locale *first-order-select* = *select select*
for $select :: ('f,\ 'v)\ atom\ clause \Rightarrow ('f,\ 'v)\ atom\ clause$
begin

abbreviation *is-grounding* :: $'f\ ground\text{-}select \Rightarrow bool$ **where**
$is\text{-}grounding\ select_G \equiv is\text{-}select\text{-}grounding\ select\ select_G$

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*
⟨*proof*⟩

lemma *select-from-ground-clause1*:
assumes *clause.is-ground clause*
shows *clause.is-ground* (*select clause*)
⟨*proof*⟩

lemma *select-from-ground-clause2*:
assumes *literal* ∈# *select* (*clause.from-ground clause*)
shows *literal.is-ground literal*
⟨*proof*⟩

lemma *select-from-ground-clause3*:
assumes *clause.is-ground clause literal*$_G$ ∈# *clause.to-ground clause*
shows *literal.from-ground literal*$_G$ ∈# *clause*
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *select-subst2*:
assumes *literal* ∈# *select clause* · γ
shows *is-neg literal*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma $select_G$*-negative*:
assumes $literal_G$ ∈# $select_G$ $clause_G$
shows *is-neg* $literal_G$
⟨*proof*⟩

sublocale *ground*: *select* $select_G$
⟨*proof*⟩

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*:
$\bigwedge context\ term_1\ term_2.$
$term_1 \prec_t term_2 \Longrightarrow$
$term.is\text{-}ground\ term_1 \Longrightarrow$
$term.is\text{-}ground\ term_2 \Longrightarrow$
$context.is\text{-}ground\ context \Longrightarrow$
$context\langle term_1\rangle \prec_t context\langle term_2\rangle$ **and**
$less_t$*-ground-subst-stability*:
$\bigwedge term_1\ term_2\ (\gamma :: 'v \Rightarrow ('f, 'v)\ term).$
$term.is\text{-}ground\ (term_1 \cdot t\ \gamma) \Longrightarrow$
$term.is\text{-}ground\ (term_2 \cdot t\ \gamma) \Longrightarrow$
$term_1 \prec_t term_2 \Longrightarrow$
$term_1 \cdot t\ \gamma \prec_t term_2 \cdot t\ \gamma$ **and**
$less_t$*-ground-subterm-property*:
$\bigwedge term_G\ context_G.$
$term.is\text{-}ground\ term_G \Longrightarrow$
$context.is\text{-}ground\ context_G \Longrightarrow$
$context_G \neq \Box \Longrightarrow$
$term_G \prec_t context_G\langle term_G\rangle$
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**
$term_{G1} \prec_{tG} term_{G2} \equiv term.from\text{-}ground\ term_{G1} \prec_t term.from\text{-}ground\ term_{G2}$

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*⟩

lemmas $less_l$*-def* = *less-lit-def*
lemmas $less_c$*-def* = *less-cls-def*

abbreviation $less\text{-}eq_l$ (**infix** $\preceq_l$ *50*) **where**
$less\text{-}eq_l \equiv (\prec_l)^{==}$

abbreviation *less-eq*$_c$ (**infix** $\preceq_c$ *50*) **where**
less-eq$_c$ $\equiv$ $(\prec_c)^{==}$

abbreviation *is-maximal*$_l$ ::
('*f*, '*v*) *atom literal* $\Rightarrow$ ('*f*, '*v*) *atom clause* $\Rightarrow$ *bool* **where**
is-maximal$_l$ *literal clause* $\equiv$ *is-maximal-in-mset-wrt* $(\prec_l)$ *clause literal*

abbreviation *is-strictly-maximal*$_l$ ::
('*f*, '*v*) *atom literal* $\Rightarrow$ ('*f*, '*v*) *atom clause* $\Rightarrow$ *bool* **where**
is-strictly-maximal$_l$ *literal clause* $\equiv$ *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*⟩

lemma *less*$_t$*-total-on'*: *totalp-on* (*term.from-ground* ‘ *terms*$_G$) $(\prec_t)$
⟨*proof*⟩

lemma *less*$_{tG}$*-wellfounded*: *wfp* $(\prec_{tG})$
⟨*proof*⟩

6.3 Ground term ordering

lemma *less*$_{tG}$*-asymmetric* [*intro*]: *asymp* $(\prec_{tG})$
⟨*proof*⟩

lemmas *less*$_{tG}$*-asymmetric-on* = *less*$_{tG}$*-asymmetric*[*THEN asymp-on-subset*, *OF subset-UNIV*]

lemma *less*$_{tG}$*-transitive* [*intro*]: *transp* $(\prec_{tG})$
⟨*proof*⟩

lemmas *less*$_{tG}$*-transitive-on* = *less*$_{tG}$*-transitive*[*THEN transp-on-subset*, *OF subset-UNIV*]

lemma *less*$_{tG}$*-total* [*intro*]: *totalp* $(\prec_{tG})$
⟨*proof*⟩

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 $context\langle term_1\rangle_G \prec_{tG} context\langle term_2\rangle_G$

⟨*proof*⟩

lemma $less_{tG}$*-subterm-property* [*simp*]:
assumes $context \neq \Box_G$
shows $term \prec_{tG} context\langle term\rangle_G$
⟨*proof*⟩

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* $term_2$
⟨*proof*⟩

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 $term_1 \cdot t\ \gamma \preceq_t term_2 \cdot t\ \gamma$
⟨*proof*⟩

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 $literal \prec_l literal' \Longrightarrow literal \cdot l\ \gamma \prec_l literal' \cdot l\ \gamma$
⟨*proof*⟩

lemma *maximal*$_l$*-in-clause*:
assumes *is-maximal*$_l$ *literal clause*
shows *literal* $\in\#$ *clause*
⟨*proof*⟩

lemma *strictly-maximal*$_l$*-in-clause*:

assumes *is-strictly-maximal*$_l$ *literal clause*
shows *literal* $\in\#$ *clause*
⟨*proof*⟩

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 $(clause \cdot \gamma)$
clause.is-ground $(clause' \cdot \gamma)$
shows $clause \prec_c clause' \Longrightarrow clause \cdot \gamma \prec_c clause' \cdot \gamma$
⟨*proof*⟩

6.6 Grounding

sublocale *ground*: *ground-ordering* $(\prec_{tG})$
⟨*proof*⟩

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*$_{tG}$: $\neg\ term_{G2} \preceq_{tG} term_{G1} \longleftrightarrow term_{G1} \prec_{tG} term_{G2}$
⟨*proof*⟩

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* $term_2$
⟨*proof*⟩

lemma *less-eq*$_{tG}$*-less-eq*$_t$:
$term_{G1} \preceq_{tG} term_{G2} \longleftrightarrow$ *term.from-ground* $term_{G1} \preceq_t$ *term.from-ground* $term_{G2}$
⟨*proof*⟩

lemma *not-less-eq*$_t$:
assumes *term.is-ground* $term_1$ **and** *term.is-ground* $term_2$
shows $\neg\ term_2 \preceq_t term_1 \longleftrightarrow term_1 \prec_t term_2$
⟨*proof*⟩

lemma *less*$_{lG}$*-less*$_l$:
$literal_{G1} \prec_{lG} literal_{G2} \longleftrightarrow$ *literal.from-ground* $literal_{G1} \prec_l$ *literal.from-ground* $literal_{G2}$

⟨*proof*⟩

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* $literal_2$
⟨*proof*⟩

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* $literal_2$
⟨*proof*⟩

lemma *less-*eq_{lG}*-less-*eq_l:
$literal_{G1} \preceq_{lG} literal_{G2} \longleftrightarrow$ *literal.from-ground* $literal_{G1} \preceq_l$ *literal.from-ground* $literal_{G2}$
⟨*proof*⟩

lemma *maximal-lit-in-clause*:
assumes *ground.is-maximal-lit* $literal_G$ $clause_G$
shows $literal_G \in\# clause_G$
⟨*proof*⟩

lemma *is-*$maximal_l$*-empty* [*simp*]:
assumes *is-*$maximal_l$ *literal* {#}
shows *False*
⟨*proof*⟩

lemma *is-strictly-*$maximal_l$*-empty* [*simp*]:
assumes *is-strictly-*$maximal_l$ *literal* {#}
shows *False*
⟨*proof*⟩

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* $clause_G$)
⟨*proof*⟩

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* $clause_G$)
⟨*proof*⟩

lemma *not-less-*eq_{lG}: $\neg\ literal_{G2} \preceq_{lG} literal_{G1} \longleftrightarrow literal_{G1} \prec_{lG} literal_{G2}$
⟨*proof*⟩

lemma *not-less-*eq_l:
assumes *literal.is-ground* $literal_1$ **and** *literal.is-ground* $literal_2$

shows ¬ $literal_2 \preceq_l literal_1 \longleftrightarrow literal_1 \prec_l literal_2$
⟨*proof*⟩

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*⟩

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* $clause_2$
⟨*proof*⟩

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* $clause_2$
⟨*proof*⟩

lemma *less-eq*$_{cG}$*-less-eq*$_c$:
$clause_{G1} \preceq_{cG} clause_{G2} \longleftrightarrow$ *clause.from-ground* $clause_{G1} \preceq_c$ *clause.from-ground* $clause_{G2}$
⟨*proof*⟩

lemma *not-less-eq*$_{cG}$: ¬ $clause_{G2} \preceq_{cG} clause_{G1} \longleftrightarrow clause_{G1} \prec_{cG} clause_{G2}$
⟨*proof*⟩

lemma *not-less-eq*$_c$:
assumes *clause.is-ground* $clause_1$ **and** *clause.is-ground* $clause_2$
shows ¬ $clause_2 \preceq_c clause_1 \longleftrightarrow clause_1 \prec_c clause_2$
⟨*proof*⟩

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′*
⟨*proof*⟩

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′*⟩ $\longleftrightarrow$ *term* $\prec_t$ *term′*

$\langle proof \rangle$

6.7 Stability under ground substitution

lemma *$less_t$-less-eq_t-ground-subst-stability*:
assumes
$term.is\text{-}ground\ (term_1 \cdot t\ \gamma)$
$term.is\text{-}ground\ (term_2 \cdot t\ \gamma)$
$term_1 \cdot t\ \gamma \prec_t term_2 \cdot t\ \gamma$
shows
$\neg\ term_2 \preceq_t term_1$
$\langle proof \rangle$

lemma *less-eq_l-ground-subst-stability*:
assumes
$literal.is\text{-}ground\ (literal_1 \cdot l\ \gamma)$
$literal.is\text{-}ground\ (literal_2 \cdot l\ \gamma)$
$literal_1 \preceq_l literal_2$
shows $literal_1 \cdot l\ \gamma \preceq_l literal_2 \cdot l\ \gamma$
$\langle proof \rangle$

lemma *$less_l$-less-eq_l-ground-subst-stability*: **assumes**
$literal.is\text{-}ground\ (literal_1 \cdot l\ \gamma)$
$literal.is\text{-}ground\ (literal_2 \cdot l\ \gamma)$
$literal_1 \cdot l\ \gamma \prec_l literal_2 \cdot l\ \gamma$
shows
$\neg\ literal_2 \preceq_l literal_1$
$\langle proof \rangle$

lemma *less-eq_c-ground-subst-stability*:
assumes
$clause.is\text{-}ground\ (clause_1 \cdot \gamma)$
$clause.is\text{-}ground\ (clause_2 \cdot \gamma)$
$clause_1 \preceq_c clause_2$
shows $clause_1 \cdot \gamma \preceq_c clause_2 \cdot \gamma$
$\langle proof \rangle$

lemma *$less_c$-less-eq_c-ground-subst-stability*: **assumes**
$clause.is\text{-}ground\ (clause_1 \cdot \gamma)$
$clause.is\text{-}ground\ (clause_2 \cdot \gamma)$
$clause_1 \cdot \gamma \prec_c clause_2 \cdot \gamma$
shows
$\neg\ clause_2 \preceq_c clause_1$
$\langle proof \rangle$

lemma *is-$maximal_l$-ground-subst-stability*:
assumes
clause-not-empty: $clause \neq \{\#\}$ **and**
clause-grounding: $clause.is\text{-}ground\ (clause \cdot \gamma)$

obtains *literal*
where *is-maximal*$_l$ *literal clause is-maximal*$_l$ (*literal* ·*l* γ) (*clause* · γ)
⟨*proof*⟩

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*⟩

lemma *less*$_l$*-total-on* [*intro*]: *totalp-on* (*literal.from-ground* ‘ *literals*$_G$) ($\prec_l$)
⟨*proof*⟩

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*) ($\prec_c$)
⟨*proof*⟩

lemma *unique-maximal-in-ground-clause*:
assumes
clause.is-ground clause
is-maximal$_l$ *literal clause*
is-maximal$_l$ *literal′ clause*
shows
literal = *literal′*
⟨*proof*⟩

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*⟩

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*⟩

lemma *is-strictly-maximal*$_l$*-ground-subst-stability'*:
  assumes
    literal $\in\#$ *clause*
    clause.is-ground (*clause* $\cdot$ γ)
    is-strictly-maximal$_l$ (*literal* $\cdot l$ γ) (*clause* $\cdot$ γ)
  shows
    is-strictly-maximal$_l$ *literal clause*
  ⟨*proof*⟩

lemma *less*$_t$*-less*$_l$:
  assumes $term_1 \prec_t term_2$
  shows
    $term_1 \approx term_3 \prec_l term_2 \approx term_3$
    $term_1 \,!\!\approx term_3 \prec_l term_2 \,!\!\approx term_3$
  ⟨*proof*⟩

lemma *less*$_t$*-less*$_l$*'*:
  assumes
    $\forall$ *term* $\in$ *set-uprod* (*atm-of literal*). *term* $\cdot t$ $\sigma' \preceq_t$ *term* $\cdot t$ σ
    $\exists$ *term* $\in$ *set-uprod* (*atm-of literal*). *term* $\cdot t$ $\sigma' \prec_t$ *term* $\cdot t$ σ
  shows *literal* $\cdot l$ $\sigma' \prec_l$ *literal* $\cdot l$ σ
⟨*proof*⟩

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* $\cdot l$ σ
  shows *clause* $\cdot$ $\sigma' \preceq_c$ *clause* $\cdot$ σ
  ⟨*proof*⟩

lemma *less*$_l$*-less*$_c$:
  assumes
    $\forall$ *literal* $\in\#$ *clause*. *literal* $\cdot l$ $\sigma' \preceq_l$ *literal* $\cdot l$ σ
    $\exists$ *literal* $\in\#$ *clause*. *literal* $\cdot l$ $\sigma' \prec_l$ *literal* $\cdot l$ σ
  shows *clause* $\cdot$ $\sigma' \prec_c$ *clause* $\cdot$ σ
  ⟨*proof*⟩

6.8 Substitution update

lemma *less*$_t$*-subst-upd*:
  fixes γ :: (*'f*, *'v*) *subst*
  assumes
    update-is-ground: *term.is-ground update* **and**
    update-less: *update* $\prec_t$ γ *var* **and**
    term-grounding: *term.is-ground* (*term* $\cdot t$ γ) **and**

var: $var \in$ *term.vars term*
shows $term \cdot t\ \gamma(var := update) \prec_t term \cdot t\ \gamma$
⟨*proof*⟩

lemma *less*$_l$*-subst-upd*:
fixes γ :: ('*f*, '*v*) *subst*
assumes
update-is-ground: *term.is-ground update* **and**
update-less: $update \prec_t \gamma\ var$ **and**
literal-grounding: *literal.is-ground* (*literal* $\cdot l\ \gamma$) **and**
var: $var \in$ *literal.vars literal*
shows $literal \cdot l\ \gamma(var := update) \prec_l literal \cdot l\ \gamma$
⟨*proof*⟩

lemma *less*$_c$*-subst-upd*:
assumes
update-is-ground: *term.is-ground update* **and**
update-less: $update \prec_t \gamma\ var$ **and**
literal-grounding: *clause.is-ground* (*clause* $\cdot\ \gamma$) **and**
var: $var \in$ *clause.vars clause*
shows $clause \cdot \gamma(var := update) \prec_c clause \cdot \gamma$
⟨*proof*⟩

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: $\bigwedge\tau$. ∃*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*| ≤*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* $clause_G$) ∧

$asymp\ (typed\text{-}tiebreakers\ clause_G)$
$\langle proof \rangle$

definition *is-merged-var-type-env* **where**
$is\text{-}merged\text{-}var\text{-}type\text{-}env\ \mathcal{V}\ X\ \mathcal{V}_X\ \varrho_X\ \ Y\ \mathcal{V}_Y\ \varrho_Y \equiv$
$(\forall x \in X.\ welltyped\ typeof\text{-}fun\ \mathcal{V}\ (\varrho_X\ \ x)\ (\mathcal{V}_X\ \ x)) \wedge$
$(\forall y \in Y.\ welltyped\ typeof\text{-}fun\ \mathcal{V}\ (\varrho_Y\ \ y)\ (\mathcal{V}_Y\ \ y))$

inductive *eq-resolution* :: $('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow ('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow$ *bool* **where**
eq-resolutionI:
$premise = add\text{-}mset\ literal\ premise' \Longrightarrow$
$literal = term\ !{\approx}\ term' \Longrightarrow$
$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{\ term,\ term'\ \}\} \Longrightarrow$
$welltyped\text{-}imgu'\ typeof\text{-}fun\ \mathcal{V}\ term\ term'\ \mu \Longrightarrow$
$select\ premise = \{\#\} \wedge is\text{-}maximal_l\ (literal\ \cdot l\ \mu)\ (premise\ \cdot\ \mu) \vee$
$is\text{-}maximal_l\ (literal\ \cdot l\ \mu)\ ((select\ premise)\ \cdot\ \mu) \Longrightarrow$
$conclusion = premise'\ \cdot\ \mu \Longrightarrow$
$eq\text{-}resolution\ (premise,\ \mathcal{V})\ (conclusion,\ \mathcal{V})$

inductive *eq-factoring* :: $('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow ('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow$ *bool* **where**
eq-factoringI:
$premise = add\text{-}mset\ literal_1\ (add\text{-}mset\ literal_2\ premise') \Longrightarrow$
$literal_1 = term_1 \approx term_1' \Longrightarrow$
$literal_2 = term_2 \approx term_2' \Longrightarrow$
$select\ premise = \{\#\} \Longrightarrow$
$is\text{-}maximal_l\ (literal_1\ \cdot l\ \mu)\ (premise\ \cdot\ \mu) \Longrightarrow$
$\neg\ (term_1\ \cdot t\ \mu \preceq_t term_1'\ \cdot t\ \mu) \Longrightarrow$
$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{\ term_1,\ term_2\ \}\} \Longrightarrow$
$welltyped\text{-}imgu'\ typeof\text{-}fun\ \mathcal{V}\ term_1\ term_2\ \mu \Longrightarrow$
$conclusion = add\text{-}mset\ (term_1 \approx term_2')\ (add\text{-}mset\ (term_1'\ !{\approx}\ term_2')\ premise')$
$\cdot\ \mu \Longrightarrow$
$eq\text{-}factoring\ (premise,\ \mathcal{V})\ (conclusion,\ \mathcal{V})$

inductive *superposition* ::
$('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow ('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow ('f,\ 'v,\ 'ty)\ typed\text{-}clause \Rightarrow$
bool
where
superpositionI:
$term\text{-}subst.is\text{-}renaming\ \varrho_1 \Longrightarrow$
$term\text{-}subst.is\text{-}renaming\ \varrho_2 \Longrightarrow$
$clause.vars\ (premise_1\ \cdot\ \varrho_1) \cap\ clause.vars\ (premise_2\ \cdot\ \varrho_2) = \{\} \Longrightarrow$
$premise_1 = add\text{-}mset\ literal_1\ premise_1' \Longrightarrow$
$premise_2 = add\text{-}mset\ literal_2\ premise_2' \Longrightarrow$
$\mathcal{P} \in \{Pos,\ Neg\} \Longrightarrow$
$literal_1 = \mathcal{P}\ (Upair\ context_1\langle term_1\rangle\ term_1') \Longrightarrow$
$literal_2 = term_2 \approx term_2' \Longrightarrow$
$\neg\ is\text{-}Var\ term_1 \Longrightarrow$

$term\text{-}subst.is\text{-}imgu\ \mu\ \{\{term_1 \cdot t\ \varrho_1,\ term_2 \cdot t\ \varrho_2\}\} \Longrightarrow$
$welltyped\text{-}imgu'\ typeof\text{-}fun\ \mathcal{V}_3\ (term_1 \cdot t\ \varrho_1)\ (term_2 \cdot t\ \varrho_2)\ \mu \Longrightarrow$
$\forall x \in clause.vars\ (premise_1 \cdot \varrho_1).\ \mathcal{V}_1\ (the\text{-}inv\ \varrho_1\ (Var\ x)) = \mathcal{V}_3\ x \Longrightarrow$
$\forall x \in clause.vars\ (premise_2 \cdot \varrho_2).\ \mathcal{V}_2\ (the\text{-}inv\ \varrho_2\ (Var\ x)) = \mathcal{V}_3\ x \Longrightarrow$
$welltyped_\sigma\text{-}on\ (clause.vars\ premise_1)\ typeof\text{-}fun\ \mathcal{V}_1\ \varrho_1 \Longrightarrow$
$welltyped_\sigma\text{-}on\ (clause.vars\ premise_2)\ typeof\text{-}fun\ \mathcal{V}_2\ \varrho_2 \Longrightarrow$
$(\bigwedge \tau\ \tau'.\ has\text{-}type\ typeof\text{-}fun\ \mathcal{V}_2\ term_2\ \tau \Longrightarrow has\text{-}type\ typeof\text{-}fun\ \mathcal{V}_2\ term_2'\ \tau'$
$\Longrightarrow \tau = \tau') \Longrightarrow$
$\neg\ (premise_1 \cdot \varrho_1 \cdot \mu \preceq_c premise_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
$(\mathcal{P} = Pos$
$\wedge\ select\ premise_1 = \{\#\}$
$\wedge\ is\text{-}strictly\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ (premise_1 \cdot \varrho_1 \cdot \mu)) \vee$
$(\mathcal{P} = Neg$
$\wedge\ (select\ premise_1 = \{\#\} \wedge is\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ (premise_1 \cdot \varrho_1 \cdot$
$\mu)$
$\vee\ is\text{-}maximal_l\ (literal_1 \cdot l\ \varrho_1 \cdot l\ \mu)\ ((select\ premise_1) \cdot \varrho_1 \cdot \mu))) \Longrightarrow$
$select\ premise_2 = \{\#\} \Longrightarrow$
$is\text{-}strictly\text{-}maximal_l\ (literal_2 \cdot l\ \varrho_2 \cdot l\ \mu)\ (premise_2 \cdot \varrho_2 \cdot \mu) \Longrightarrow$
$\neg\ (context_1\langle term_1\rangle \cdot t\ \varrho_1 \cdot t\ \mu \preceq_t term_1' \cdot t\ \varrho_1 \cdot t\ \mu) \Longrightarrow$
$\neg\ (term_2 \cdot t\ \varrho_2 \cdot t\ \mu \preceq_t term_2' \cdot t\ \varrho_2 \cdot t\ \mu) \Longrightarrow$
$conclusion = add\text{-}mset\ (\mathcal{P}\ (Upair\ (context_1 \cdot t_c\ \varrho_1)\langle term_2' \cdot t\ \varrho_2\rangle\ (term_1' \cdot t\ \varrho_1)))$

$(premise_1' \cdot \varrho_1 + premise_2' \cdot \varrho_2) \cdot \mu \Longrightarrow$
$all\text{-}types\ \mathcal{V}_1 \Longrightarrow all\text{-}types\ \mathcal{V}_2 \Longrightarrow$
$superposition\ (premise_2,\ \mathcal{V}_2)\ (premise_1,\ \mathcal{V}_1)\ (conclusion,\ \mathcal{V}_3)$

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**
$superposition\text{-}inferences \equiv \{\ Infer\ [premise_2,\ premise_1]\ conclusion$
$\mid\ premise_2\ premise_1\ conclusion.\ superposition\ premise_2\ premise_1\ conclusion\}$

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**
$bottom_F \equiv \{(\{\#\},\ \mathcal{V})\ \mid\ \mathcal{V}.\ all\text{-}types\ \mathcal{V}\ \}$

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 $\varrho_1 \Longrightarrow$
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 $P_1 = \{\#\} \Longrightarrow$
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
$\langle proof\rangle$

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 $P_1 = \{\#\} \land$
is-maximal$_l$ $(L_1 \cdot l\ \varrho_1 \cdot l\ \mu)$ $(P_1 \cdot \varrho_1 \cdot \mu) \lor$ *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* $P_2\ P_1\ C$
⟨*proof*⟩

lemma *superposition-iff-pos-or-neg*:
superposition $P_2\ P_1\ C \longleftrightarrow$ *pos-superposition* $P_2\ P_1\ C \lor$ *neg-superposition* $P_2\ P_1\ C$
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

lemma *welltyped-has-type*:
assumes *welltyped typeof-fun* $\mathcal{V}$ *term* τ
shows *has-type typeof-fun* $\mathcal{V}$ *term* τ
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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* = $select_G$
⟨*proof*⟩

definition *is-inference-grounding* **where**
$is\text{-}inference\text{-}grounding\ \iota\ \iota_G\ \gamma\ \varrho_1\ \varrho_2 \equiv$
$(case\ \iota\ of$
$Infer\ [(premise, \mathcal{V}')]\ (conclusion, \mathcal{V}) \Rightarrow$
$term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma$
$\wedge\ \iota_G = Infer\ [clause.to\text{-}ground\ (premise \cdot \gamma)]\ (clause.to\text{-}ground\ (conclusion \cdot \gamma))$
$\wedge\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}\ premise$
$\wedge\ welltyped_\sigma\text{-}on\ (clause.vars\ conclusion)\ typeof\text{-}fun\ \mathcal{V}\ \gamma$
$\wedge\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}\ conclusion$
$\wedge\ \mathcal{V} = \mathcal{V}'$
$\wedge\ all\text{-}types\ \mathcal{V}$
$|\ Infer\ [(premise_2, \mathcal{V}_2), (premise_1, \mathcal{V}_1)]\ (conclusion, \mathcal{V}_3) \Rightarrow$
$term\text{-}subst.is\text{-}renaming\ \varrho_1$
$\wedge\ term\text{-}subst.is\text{-}renaming\ \varrho_2$
$\wedge\ clause.vars\ (premise_1 \cdot \varrho_1) \cap clause.vars\ (premise_2 \cdot \varrho_2) = \{\}$
$\wedge\ term\text{-}subst.is\text{-}ground\text{-}subst\ \gamma$
$\wedge\ \iota_G =$
$Infer$
$[clause.to\text{-}ground\ (premise_2 \cdot \varrho_2 \cdot \gamma),\ clause.to\text{-}ground\ (premise_1 \cdot \varrho_1 \cdot \gamma)]$
$(clause.to\text{-}ground\ (conclusion \cdot \gamma))$
$\wedge\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_1\ premise_1$
$\wedge\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_2\ premise_2$
$\wedge\ welltyped_\sigma\text{-}on\ (clause.vars\ conclusion)\ typeof\text{-}fun\ \mathcal{V}_3\ \gamma$
$\wedge\ welltyped_c\ typeof\text{-}fun\ \mathcal{V}_3\ conclusion$
$\wedge\ all\text{-}types\ \mathcal{V}_1 \wedge all\text{-}types\ \mathcal{V}_2 \wedge all\text{-}types\ \mathcal{V}_3$
$|\ \text{-} \Rightarrow False$
$)$
$\wedge\ \iota_G \in ground.G\text{-}Inf$

definition *inference-groundings* **where**
$inference\text{-}groundings\ \iota = \{\ \iota_G\ |\ \iota_G\ \gamma\ \varrho_1\ \varrho_2.\ is\text{-}inference\text{-}grounding\ \iota\ \iota_G\ \gamma\ \varrho_1\ \varrho_2\ \}$

lemma *is-inference-grounding-inference-groundings*:
$is\text{-}inference\text{-}grounding\ \iota\ \iota_G\ \gamma\ \varrho_1\ \varrho_2 \Longrightarrow \iota_G \in inference\text{-}groundings\ \iota$
⟨*proof*⟩

lemma *inference*$_G$*-concl-in-clause-grounding*:
assumes $\iota_G \in inference\text{-}groundings\ \iota$
shows $concl\text{-}of\ \iota_G \in clause\text{-}groundings\ typeof\text{-}fun\ (concl\text{-}of\ \iota)$
⟨*proof*⟩

lemma *inference*$_G$*-red-in-clause-grounding-of-concl*:
assumes $\iota_G \in inference\text{-}groundings\ \iota$
shows $\iota_G \in ground.Red\text{-}I\ (clause\text{-}groundings\ typeof\text{-}fun\ (concl\text{-}of\ \iota))$
⟨*proof*⟩

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*⟩

lemma *welltyped*$_\sigma$*-on-empty*: *welltyped*$_\sigma$*-on* {} $\mathcal{F}$ $\mathcal{V}$ σ
  ⟨*proof*⟩

sublocale *lifting*:
    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* ∘ *inference-groundings*)
        typed-tiebreakers
⟨*proof*⟩

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*
    λ*select*$_G$. *Some* ∘
      (*grounded-first-order-superposition-calculus.inference-groundings* ($\prec_t$) *select*$_G$
typeof-fun)
    typed-tiebreakers
⟨*proof*⟩

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* τ $\longleftrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*term* $\cdot t$ γ) τ
⟨*proof*⟩

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
⟨*proof*⟩

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*$_G$ ≡ *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-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′ $\cdot$ γ = *conclusion* $\cdot$ γ
⟨*proof*⟩

lemma *if-subst-sth* [*clause-simp*]: (*if b then Pos else Neg*) *atom* $\cdot l$ ϱ =
(*if b then Pos else Neg*) (*atom* $\cdot a$ ϱ)
⟨*proof*⟩

lemma *superposition-lifting*:
fixes
$premise_{G1}$ $premise_{G2}$ $conclusion_G$ :: ′*f gatom clause* **and**
$premise_1$ $premise_2$ *conclusion* :: (′*f*, ′*v*) *atom clause* **and**
γ ϱ_1 ϱ_2 :: (′*f*, ′*v*) *subst* **and**
$\mathcal{V}_1$ $\mathcal{V}_2$
defines
$premise_{G1}$ [*simp*]: $premise_{G1}$ $\equiv$ *clause.to-ground* ($premise_1$ $\cdot$ ϱ_1 $\cdot$ γ) **and**
$premise_{G2}$ [*simp*]: $premise_{G2}$ $\equiv$ *clause.to-ground* ($premise_2$ $\cdot$ ϱ_2 $\cdot$ γ) **and**
$conclusion_G$ [*simp*]: $conclusion_G$ $\equiv$ *clause.to-ground* (*conclusion* $\cdot$ γ) **and**
premise-groundings [*simp*]:
premise-groundings $\equiv$ *clause-groundings typeof-fun* ($premise_1$, $\mathcal{V}_1$) $\cup$
clause-groundings typeof-fun ($premise_2$, $\mathcal{V}_2$) **and**
ι_G [*simp*]: ι_G $\equiv$ *Infer* [$premise_{G2}$, $premise_{G1}$] $conclusion_G$
assumes
renaming:
term-subst.is-renaming ϱ_1
term-subst.is-renaming ϱ_2
clause.vars ($premise_1$ $\cdot$ ϱ_1) $\cap$ *clause.vars* ($premise_2$ $\cdot$ ϱ_2) = {} **and**
$premise_1$*-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**
conclusion-grounding: *clause.is-ground* (*conclusion* $\cdot$ γ) **and**
select:
clause.from-ground ($select_G$ $premise_{G1}$) = (*select* $premise_1$) $\cdot$ ϱ_1 $\cdot$ γ
clause.from-ground ($select_G$ $premise_{G2}$) = (*select* $premise_2$) $\cdot$ ϱ_2 $\cdot$ γ **and**
ground-superposition: *ground.ground-superposition* $premise_{G2}$ $premise_{G1}$ $conclusion_G$ **and**
non-redundant: ι_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* $\cdot\ \gamma$
⟨*proof*⟩

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*⟩

lemma *eq-factoring-ground-instance*:
assumes
$\iota_G \in$ *ground.eq-factoring-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*⟩

lemma *subst-compose-if*: $\sigma \odot$ (λ*var. if var* $\in$ *range-vars'* σ *then* σ_1 *var else* σ_2 *var*) $= \sigma \odot \sigma_1$
⟨*proof*⟩

lemma *subst-compose-if'*:
assumes *range-vars'* $\sigma \cap$ *range-vars'* $\sigma' = \{\}$
shows $\sigma \odot$ (λ*var. if var* $\in$ *range-vars'* σ' *then* σ_1 *var else* σ_2 *var*) $= \sigma \odot \sigma_2$
⟨*proof*⟩

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* (λ*var. if b var then* γ_1 *var else* γ_2 *var*)
⟨*proof*⟩

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*⟩

lemma *ground-instances*:
assumes
$\iota_G \in$ *ground.Inf-from-q select*$_G$ ($\bigcup$ (*clause-groundings typeof-fun* ‘ *premises*))
$\iota_G \notin$ *ground.Red-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*⟩

end

context *first-order-superposition-calculus*
begin

lemma *overapproximation*:
obtains *select*$_G$ **where**
ground-Inf-overapproximated select$_G$ *premises*
is-grounding select$_G$
⟨*proof*⟩

sublocale *statically-complete-calculus* $\bot_F$ *inferences entails-$\mathcal{G}$ Red-I-$\mathcal{G}$ Red-F-$\mathcal{G}$*
⟨*proof*⟩

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
⟨*proof*⟩
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*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *less-kbo-subst*:
fixes *σ* :: (′*f* :: *weighted*, ′*v*) *subst*
shows *less-kbo s t* ⟹ *less-kbo* (*s* · *σ*) (*t* · *σ*)
⟨*proof*⟩

lemma *wfP-less-kbo*: *wfP less-kbo*
⟨*proof*⟩

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*:
⋀(*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*⟩

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* $\mathcal{V}$ γ
  obtains γ'
  where
    term-subst.is-ground-subst γ'
    $welltyped_\sigma$ *typeof-fun* $\mathcal{V}$ γ'
    $\forall x \in$ *clause.vars* C. $\gamma\ x = \gamma'\ x$
  ⟨*proof*⟩

lemma *vars-subst*: $\bigcup$ (*term.vars* ‘ ϱ ‘ *term.vars* t) = *term.vars* $(t \cdot t\ \varrho)$
  ⟨*proof*⟩

lemma $vars\text{-}subst_a$: $\bigcup$ (*term.vars* ‘ ϱ ‘ *atom.vars* a) = *atom.vars* $(a \cdot a\ \varrho)$
  ⟨*proof*⟩

lemma $vars\text{-}subst_l$: $\bigcup$ (*term.vars* ‘ ϱ ‘ *literal.vars* l) = *literal.vars* $(l \cdot l\ \varrho)$
  ⟨*proof*⟩

lemma $vars\text{-}subst_c$: $\bigcup$ (*term.vars* ‘ ϱ ‘ *clause.vars* C) = *clause.vars* $(C \cdot \varrho)$
  ⟨*proof*⟩

lemma *eq-resolution-sound*:
  assumes *step*: *eq-resolution* P C
  shows $\{P\} \models_F \{C\}$
  ⟨*proof*⟩

lemma *eq-factoring-sound*:
  assumes *step*: *eq-factoring* P C
  shows $\{P\} \models_F \{C\}$
  ⟨*proof*⟩

lemma *superposition-sound*:
  assumes *step*: *superposition P2 P1 C*
  shows $\{P1, P2\} \models_F \{C\}$
  ⟨*proof*⟩

end

sublocale *grounded-first-order-superposition-calculus* ⊆
  sound-inference-system inferences $\bot_F$ ($\models_F$)
⟨*proof*⟩

sublocale *first-order-superposition-calculus* ⊆
  sound-inference-system inferences $\bot_F$ *entails-*$\mathcal{G}$
⟨*proof*⟩

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* $\{P1, P2\}$ $\{C\}$
  ⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *soundness-ground-eq-resolution*:
  assumes *step*: *ground-eq-resolution P C*
  shows *G-entails* $\{P\}$ $\{C\}$
  ⟨*proof*⟩

lemma (**in** *ground-superposition-calculus*) *soundness-ground-eq-factoring*:
  assumes *step*: *ground-eq-factoring P C*
  shows *G-entails* $\{P\}$ $\{C\}$
  ⟨*proof*⟩

sublocale *ground-superposition-calculus* ⊆ *sound-inference-system* **where**
  Inf = *G-Inf* **and**
  Bot = *G-Bot* **and**
  entails = *G-entails*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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.