First Order Clause

Balazs Toth

March 20, 2025

Abstract

This entry provides reusable theories that lift properties of first-order (ground and nonground) terms to atoms, literals, and clauses. These properties include substitutions, orders, entailment, and typing. The sessions `AFP/First_Order_Terms` and `AFP/Abstract_Substitution` are the basis of this entry.

Contents

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

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

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

end
theory *Ground-Context*
imports *Ground-Term-Extra*
begin

type-synonym *'f ground-context* = (*'f*, *'f gterm*) *actxt*

abbreviation (*input*) *GHole* (‹$\square_G$›) **where**
$\square_G \equiv \square$

abbreviation *ctxt-apply-gterm* (‹-⟨-⟩$_G$› [*1000*, *0*] *1000*) **where**
$C\langle s\rangle_G \equiv GFun\langle C;s\rangle$

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

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

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

end
theory *Multiset-Extra*
imports
HOL−Library.Multiset
HOL−Library.Multiset-Order
Nested-Multisets-Ordinals.Multiset-More
Abstract-Substitution.Natural-Magma-Functor
begin

lemma *exists-multiset* [*intro*]: $\exists\, M.\ x \in \mathit{set\text{-}mset}\ M$
⟨*proof*⟩

global-interpretation *muliset-magma*: *natural-magma-with-empty* **where**
to-set = *set-mset* **and** *plus* = (+) **and** *wrap* = $\lambda l.\ \{\#l\#\}$ **and** *add* = *add-mset* **and** *empty* = $\{\#\}$
⟨*proof*⟩

global-interpretation *multiset-functor*: *finite-natural-functor* **where**
map = *image-mset* **and** *to-set* = *set-mset*
⟨*proof*⟩

global-interpretation *multiset-functor*: *natural-functor-conversion* **where**
map = *image-mset* **and** *to-set* = *set-mset* **and** *map-to* = *image-mset* **and** *map-from* = *image-mset* **and**
map′ = *image-mset* **and** *to-set′* = *set-mset*
⟨*proof*⟩

global-interpretation *muliset-functor*: *natural-magma-functor* **where**
map = *image-mset* **and** *to-set* = *set-mset* **and** *plus* = (+) **and** *wrap* = $\lambda l.\ \{\#l\#\}$ **and** *add* = *add-mset*
⟨*proof*⟩

lemma *one-le-countE*:
assumes $1 \leq \mathit{count}\ M\ x$
obtains M' **where** $M = \mathit{add\text{-}mset}\ x\ M'$
⟨*proof*⟩

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

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

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

lemma *Uniq-antimono'*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x) \Longrightarrow Uniq\ P \Longrightarrow Uniq\ Q$
⟨*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 $J \equiv B - A$ **and** $K \equiv A - B$
shows $J \neq \{\#\}$ **and** $\forall\, k \in\# K.\ \exists\, x \in\# J.\ R\ k\ x$
⟨*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*: $M \neq \{\#\} \Longrightarrow$ *multp R M* $(M + M)$
⟨*proof*⟩

lemma *mult1-implies-one-step-strong*:
assumes *trans r* **and** *asym r* **and** $(A, B) \in$ *mult1 r*
shows $B - A \neq \{\#\}$ **and** $\forall k \in\# A - B.\ \exists j \in\# B - A.\ (k, j) \in r$
⟨*proof*⟩

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* [*simp*]:
assumes *asymp R transp R*
shows *multp R* (*add-mset x X*) (*add-mset x Y*) ⟷ *multp R X Y*
⟨*proof*⟩

lemma *inj-mset-plus-same*: *inj* (λ*X* :: *'a multiset* . *X* + *X*)
⟨*proof*⟩

lemma *multp-image-lesseq-if-all-lesseq*:
assumes
asymp: *asymp R* **and**
transp: *transp R* **and**
all-lesseq: ∀ *x*∈#*X*. *R*$^{==}$ (*f x*) (*g x*)
shows (*multp R*)$^{==}$ (*image-mset f X*) (*image-mset g X*)
⟨*proof*⟩

lemma *multp-image-less-if-all-lesseq-ex-less*:
assumes
asymp: *asymp R* **and**
transp: *transp R* **and**
all-less-eq: ∀ *x*∈#*X*. *R*$^{==}$ (*f x*) (*g x*) **and**
ex-less: ∃ *x*∈#*X*. *R* (*f x*) (*g x*)
shows *multp R* {# *f x*. *x* ∈# *X* #} {# *g x*. *x* ∈# *X* #}
⟨*proof*⟩

lemma *not-reflp-multp*$_{DM}$: ¬ *reflp* (*multp*$_{DM}$ *R*)
⟨*proof*⟩

lemma *not-less-empty-multp*$_{DM}$: ¬ *multp*$_{DM}$ *R X* {#}
⟨*proof*⟩

lemma *not-reflp-multp*$_{HO}$: ¬ *reflp* (*multp*$_{HO}$ *R*)
⟨*proof*⟩

lemma *not-less-empty-multp*$_{HO}$: ¬ *multp*$_{HO}$ *R X* {#}
⟨*proof*⟩

lemma *not-refl-mult*: ¬ *refl* (*mult R*)
⟨*proof*⟩

lemma *not-less-empty-mult*: $(X, \{\#\}) \notin mult\ R$
⟨*proof*⟩

lemma *empty-less-mult*: $X \neq \{\#\} \Longrightarrow (\{\#\}, X) \in mult\ R$
⟨*proof*⟩

lemma *not-reflp-multp*: $\neg\ reflp\ (multp\ R)$
⟨*proof*⟩

lemma *empty-less-multp*: $X \neq \{\#\} \Longrightarrow multp\ R\ \{\#\}\ X$
⟨*proof*⟩

lemma *not-less-empty-multp*: $\neg\ multp\ R\ X\ \{\#\}$
⟨*proof*⟩

end
theory *Uprod-Extra*
imports
HOL−Library.Uprod
Multiset-Extra
Abstract-Substitution.Natural-Functor
begin

abbreviation *upair* **where**
$upair \equiv \lambda(x, y).\ Upair\ x\ y$

lemma *Upair-sym*: $Upair\ x\ y = Upair\ y\ x$
⟨*proof*⟩

lemma *upair-in-sym* [*simp*]:
assumes *sym I*
shows $Upair\ a\ b \in upair\ `\ I \longleftrightarrow (a, b) \in I \wedge (b, a) \in I$
⟨*proof*⟩

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

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

lemma *Abs-commute-inverse-mset* [*simp*]:
$apply\text{-}commute\ (Abs\text{-}commute\ (\lambda x\ y.\ \{\#x, y\#\})) = (\lambda x\ y.\ \{\#x, y\#\})$
⟨*proof*⟩

lemma *set-mset-mset-uprod* [*simp*]: $set\text{-}mset\ (mset\text{-}uprod\ up) = set\text{-}uprod\ up$
⟨*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*⟩

lemma *ball-set-uprod* [*simp*]: (∀ *t*∈*set-uprod* (*Upair t*$_1$ *t*$_2$). *P t*) ⟷ *P t*$_1$ ∧ *P t*$_2$
⟨*proof*⟩

lemma *inj-mset-uprod*: *inj mset-uprod*
⟨*proof*⟩

lemma *mset-uprod-plus-neq*: *mset-uprod a* ≠ *mset-uprod b* + *mset-uprod b*
⟨*proof*⟩

lemma *set-uprod-not-empty*: *set-uprod a* ≠ {}
⟨*proof*⟩

lemma *exists-uprod* [*intro*]: ∃ *a*. *x* ∈ *set-uprod a*
⟨*proof*⟩

global-interpretation *uprod-functor*: *finite-natural-functor* **where** *map* = *map-uprod* **and** *to-set* = *set-uprod*
⟨*proof*⟩

global-interpretation *uprod-functor*: *natural-functor-conversion* **where**
map = *map-uprod* **and** *to-set* = *set-uprod* **and** *map-to* = *map-uprod* **and** *map-from* = *map-uprod* **and**
map′ = *map-uprod* **and** *to-set*′ = *set-uprod*
⟨*proof*⟩

end
theory *Ground-Clause*
imports
Saturation-Framework-Extensions.Clausal-Calculus
Ground-Term-Extra
Ground-Context
Uprod-Extra
begin

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

end

theory *Typing*
imports *Main*
begin

locale *typing* =
fixes *welltyped* :: *'expr* ⇒ *'ty* ⇒ *bool*
assumes *right-unique*: *right-unique welltyped*
begin

lemmas *right-uniqueD* [*dest*] = *right-uniqueD*[*OF right-unique*]

end

locale *base-typing* = *typing*
begin

abbreviation *is-welltyped* **where**
is-welltyped expr ≡ ∃ *τ*. *welltyped expr τ*

end

locale *typing-lifting* = *sub*: *typing* **where** *welltyped* = *sub-welltyped*
for *sub-welltyped* :: *'sub* ⇒ *'ty* ⇒ *bool* +
fixes *to-set* :: *'expr* ⇒ *'sub set*
begin

definition *is-welltyped* **where**
is-welltyped expr ≡ ∃ *τ*. ∀ *sub* ∈ *to-set expr*. *sub-welltyped sub τ*

abbreviation *welltyped* **where**
welltyped expr (-::*unit*) ≡ *is-welltyped expr*

sublocale *typing* **where** *welltyped* = *welltyped*
⟨*proof*⟩

end

locale *typing-lifting′* =
fixes
sub-welltyped :: *'extra* ⇒ *'sub* ⇒ *'ty′* ⇒ *bool* **and**
to-set :: *'expr* ⇒ *'sub set*
assumes *lifting*: ⋀*extra*. *typing-lifting* (*sub-welltyped extra*)
begin

sublocale *typing-lifting* **where**
sub-welltyped = *sub-welltyped* 𝒱 **and** *to-set* = *to-set*
⟨*proof*⟩

end

end
theory *Natural-Magma-Typing-Lifting*
imports
Abstract-Substitution.Natural-Magma
Typing
begin

locale *natural-magma-is-typed-lifting* =
natural-magma **where** *to-set* = *to-set* +
typing-lifting **where** *to-set* = *to-set* **and** *sub-welltyped* = *sub-welltyped*
for *to-set* :: *'expr* ⇒ *'sub set* **and** *sub-welltyped* :: *'sub* ⇒ *unit* ⇒ *bool*
begin

abbreviation (*input*) *sub-is-welltyped* **where**
sub-is-welltyped expr ≡ *sub-welltyped expr* ()

lemma *add* [*simp*]: *is-welltyped* (*add sub M*) ⟷ (*sub-is-welltyped sub* ∧ *is-welltyped M*)
⟨*proof*⟩

lemma *plus* [*simp*]:
is-welltyped (*plus M M'*) ⟷ *is-welltyped M* ∧ *is-welltyped M'*
⟨*proof*⟩

end

locale *natural-magma-with-empty-is-typed-lifting* =
natural-magma-is-typed-lifting + *natural-magma-with-empty*
begin

lemma *empty* [*intro*]: *is-welltyped empty*
⟨*proof*⟩

end

locale *natural-magma-typing-lifting* =
welltyped: *natural-magma-is-typed-lifting* **where** *sub-welltyped* = *sub-welltyped* +
typing-lifting +
natural-magma

locale *natural-magma-with-empty-typing-lifting* =
natural-magma-typing-lifting +
welltyped: *natural-magma-with-empty-is-typed-lifting* **where** *sub-welltyped* = *sub-welltyped*

end
theory *Multiset-Typing-Lifting*
imports
Natural-Magma-Typing-Lifting

Multiset-Extra
Abstract-Substitution.Functional-Substitution-Lifting
begin

locale *mulitset-typing-lifting* =
typing-lifting **where**
to-set = *set-mset* **and** *sub-welltyped* = *sub-welltyped*
for *sub-welltyped* :: *'sub* ⇒ *unit* ⇒ *bool*
begin

sublocale *natural-magma-with-empty-typing-lifting* **where**
to-set = *set-mset* **and** *plus* = (+) **and** *wrap* = λ*l*. {#*l*#} **and** *add* = *add-mset* **and** *empty* = {#}
⟨*proof*⟩

end

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

lemma *literal-cases*: ⟦$\mathcal{P} \in \{Pos, Neg\}$; $\mathcal{P} = Pos \Longrightarrow P$; $\mathcal{P} = Neg \Longrightarrow P$⟧ ⟹ *P*
⟨*proof*⟩

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

lemma *map-literal-comp*:
map-literal f (*map-literal g l*) = *map-literal* (λ*a*. *f* (*g a*)) *l*
⟨*proof*⟩

lemma *literals-distinct* [*simp*]: *Pos* ≠ *Neg Neg* ≠ *Pos*
⟨*proof*⟩

primrec *mset-lit* :: *'a uprod literal* ⇒ *'a multiset* **where**
mset-lit (*Pos a*) = *mset-uprod a* |
mset-lit (*Neg a*) = *mset-uprod a* + *mset-uprod a*

lemma *mset-lit-image-mset*: *mset-lit* (*map-literal* (*map-uprod f*) *l*) = *image-mset f* (*mset-lit l*)
⟨*proof*⟩

lemma *uprod-mem-image-iff-prod-mem*[*simp*]:
assumes *sym I*
shows (*Upair t t'*) ∈ (λ(t_1, t_2). *Upair* t_1 t_2) ‘ *I* ⟷ (*t*, *t'*) ∈ *I*

⟨*proof*⟩

lemma *true-lit-uprod-iff-true-lit-prod*[*simp*]:
assumes *sym I*
shows
upair ‘ I ⊨*l Pos* (*Upair t t′*) ⟷ *I* ⊨*l Pos* (*t*, *t′*)
upair ‘ I ⊨*l Neg* (*Upair t t′*) ⟷ *I* ⊨*l Neg* (*t*, *t′*)
⟨*proof*⟩

abbreviation *Pos-Upair* (**infix** ≈ *66*) **where**
Pos-Upair t t′ ≡ *Pos* (*Upair t t′*)

abbreviation *Neg-Upair* (**infix** !≈ *66*) **where**
Neg-Upair t t′ ≡ *Neg* (*Upair t t′*)

lemma *exists-literal-for-atom* [*intro*]: ∃ *l*. *a* ∈ *set-literal l*
⟨*proof*⟩

lemma *exists-literal-for-term* [*intro*]: ∃ *l*. *t* ∈# *mset-lit l*
⟨*proof*⟩

lemma *finite-set-literal* [*intro*]: *finite* (*set-literal l*)
⟨*proof*⟩

lemma *map-literal-map-uprod-cong*:
assumes ⋀*t*. *t* ∈# *mset-lit l* ⟹ *f t* = *g t*
shows *map-literal* (*map-uprod f*) *l* = *map-literal* (*map-uprod g*) *l*
⟨*proof*⟩

lemma *set-mset-set-uprod*: *set-mset* (*mset-lit l*) = *set-uprod* (*atm-of l*)
⟨*proof*⟩

lemma *mset-lit-set-literal*: *t* ∈# *mset-lit l* ⟷ *t* ∈ ⋃(*set-uprod ‘ set-literal l*)
⟨*proof*⟩

lemma *inj-mset-lit*: *inj mset-lit*
⟨*proof*⟩

global-interpretation *literal-functor*: *finite-natural-functor* **where**
map = *map-literal* **and** *to-set* = *set-literal*
⟨*proof*⟩

global-interpretation *literal-functor*: *natural-functor-conversion* **where**
map = *map-literal* **and** *to-set* = *set-literal* **and** *map-to* = *map-literal* **and** *map-from* = *map-literal* **and**
map′ = *map-literal* **and** *to-set′* = *set-literal*
⟨*proof*⟩

abbreviation *uprod-literal-to-set* **where** *uprod-literal-to-set l* ≡ *set-mset* (*mset-lit*

l)

abbreviation *map-uprod-literal* **where** *map-uprod-literal f* ≡ *map-literal* (*map-uprod f*)

global-interpretation *uprod-literal-functor*: *finite-natural-functor* **where**
map = *map-uprod-literal* **and** *to-set* = *uprod-literal-to-set*
⟨*proof*⟩

global-interpretation *uprod-literal-functor*: *natural-functor-conversion* **where**
map = *map-uprod-literal* **and** *to-set* = *uprod-literal-to-set* **and** *map-to* = *map-uprod-literal* **and**
map-from = *map-uprod-literal* **and** *map′* = *map-uprod-literal* **and** *to-set′* = *uprod-literal-to-set*
⟨*proof*⟩

lemma *exists-inference* [*intro*]: ∃ *ι*. *f* ∈ *set-inference ι*
⟨*proof*⟩

lemma *finite-set-inference* [*intro*]: *finite* (*set-inference ι*)
⟨*proof*⟩

global-interpretation *inference-functor*: *finite-natural-functor* **where**
map = *map-inference* **and** *to-set* = *set-inference*
⟨*proof*⟩

global-interpretation *inference-functor*: *natural-functor-conversion* **where**
map = *map-inference* **and** *to-set* = *set-inference* **and** *map-to* = *map-inference* **and**
map-from = *map-inference* **and** *map′* = *map-inference* **and** *to-set′* = *set-inference*
⟨*proof*⟩

end
theory *Clause-Typing*
imports
Multiset-Typing-Lifting

Clausal-Calculus-Extra
Multiset-Extra
Uprod-Extra
begin

locale *clause-typing* = *term*: *typing term-welltyped*
for *term-welltyped*
begin

sublocale *atom*: *typing-lifting* **where**
sub-welltyped = *term-welltyped* **and** *to-set* = *set-uprod*
⟨*proof*⟩

lemma *atom-is-welltyped-iff* [*simp*]:
atom.is-welltyped (*Upair t t'*) ⟷ (∃ τ. *term-welltyped t* τ ∧ *term-welltyped t'* τ)
⟨*proof*⟩

sublocale *literal*: *typing-lifting* **where**
sub-welltyped = *atom.welltyped* **and** *to-set* = *set-literal*
⟨*proof*⟩

lemma *literal-is-welltyped-iff* [*simp*]:
literal.is-welltyped (*t* ≈ *t'*) ⟷ *atom.is-welltyped* (*Upair t t'*)
literal.is-welltyped (*t* !≈ *t'*) ⟷ *atom.is-welltyped* (*Upair t t'*)
⟨*proof*⟩

lemma *literal-is-welltyped-iff-atm-of*:
literal.is-welltyped l ⟷ *atom.is-welltyped* (*atm-of l*)
⟨*proof*⟩

sublocale *clause*: *mulitset-typing-lifting* **where**
sub-welltyped = *literal.welltyped*
⟨*proof*⟩

end

end
theory *Context-Extra*
imports *First-Order-Terms.Subterm-and-Context*
begin

no-notation *subst-compose* (**infixl** ∘$_s$ *75*)
no-notation *subst-apply-term* (**infixl** · *67*)

end
theory *Term-Typing*
imports *Typing Context-Extra*
begin

type-synonym ('*f*, '*ty*) *fun-types* = '*f* ⇒ *nat* ⇒ '*ty list* × '*ty*

locale *context-compatible-typing* =
fixes *Fun welltyped*
assumes
context-compatible [*intro*]:
⋀*t t' c* τ τ'.
welltyped t τ' ⟹
welltyped t' τ' ⟹
welltyped (*Fun*⟨*c*; *t*⟩) τ ⟹
welltyped (*Fun*⟨*c*; *t'*⟩) τ

locale *subterm-typing* =
  fixes *Fun welltyped*
  assumes
    subterm': $\bigwedge f\ ts\ \tau.\ welltyped\ (Fun\ f\ ts)\ \tau \Longrightarrow \forall\, t{\in}set\ ts.\ \exists\, \tau'.\ welltyped\ t\ \tau'$
begin

lemma *subterm*: $welltyped\ (Fun\langle c;\ t\rangle)\ \tau \Longrightarrow \exists\, \tau.\ welltyped\ t\ \tau$
⟨*proof*⟩

end

locale *term-typing* =
  base-typing +
  welltyped: *context-compatible-typing* **where** *welltyped* = *welltyped* +
  welltyped: *subterm-typing* **where** *welltyped* = *welltyped*

end
theory *Ground-Typing*
  imports
    Ground-Clause
    Clause-Typing
    Term-Typing
begin

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

lemma *right-unique-welltyped*[*iff*]: $right\text{-}unique\ (welltyped\ \mathcal{F})$
⟨*proof*⟩

lemma *welltyped-context-compatible*:
  assumes
    t-type: $welltyped\ \mathcal{F}\ t\ \tau'$ **and**
    t'-type: $welltyped\ \mathcal{F}\ t'\ \tau'$ **and**
    c-type: $welltyped\ \mathcal{F}\ c\langle t\rangle_G\ \tau$
  shows $welltyped\ \mathcal{F}\ c\langle t'\rangle_G\ \tau$
⟨*proof*⟩

lemma *welltyped-subterms*:
  fixes $f\ ts\ \tau$
  assumes $welltyped\ \mathcal{F}\ (GFun\ f\ ts)\ \tau$
  shows $\forall\, t{\in}set\ ts.\ \exists\, \tau'.\ welltyped\ \mathcal{F}\ t\ \tau'$
  ⟨*proof*⟩

global-interpretation *term*: *term-typing* **where**
  $welltyped = welltyped\ \mathcal{F}$ **and** $Fun = GFun$
  for $\mathcal{F} :: ('f,\ 'ty)\ fun\text{-}types$
  ⟨*proof*⟩

global-interpretation *clause-typing* **where**
term-welltyped = *welltyped* $\mathcal{F}$
for $\mathcal{F}$ :: (*'f*, *'ty*) *fun-types*
⟨*proof*⟩

end
theory *Nonground-Term*
imports
Abstract-Substitution.Substitution-First-Order-Term
Abstract-Substitution.Functional-Substitution-Lifting
Ground-Term-Extra
begin

no-notation *subst-compose* (**infixl** $\circ_s$ *75*)
notation *subst-compose* (**infixl** $\odot$ *75*)

no-notation *subst-apply-term* (**infixl** · *67*)
notation *subst-apply-term* (**infixl** ·*t* *67*)

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

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

1 Nonground Terms and Substitutions

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

1.1 Unified naming

locale *vars-def* =
fixes *vars-def* :: *'expr* ⇒ *'var*
begin

abbreviation *vars* ≡ *vars-def*

end

locale *grounding-def* =
fixes
to-ground-def :: *'expr* ⇒ *'expr*$_G$ **and**
from-ground-def :: *'expr*$_G$ ⇒ *'expr*
begin

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

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

end

1.2 Term

locale *nonground-term-properties* =
base-functional-substitution +
finite-variables +
all-subst-ident-iff-ground

locale *term-grounding* =
variables-in-base-imgu **where** *base-vars* = *vars* **and** *base-subst* = *subst* +
grounding

locale *nonground-term*
begin

sublocale *vars-def* **where** *vars-def* = *vars-term* ⟨*proof*⟩

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

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

sublocale *nonground-term-properties* **where**
subst = (·*t*) **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
vars = *vars* :: ('*f*, '*v*) *term* ⇒ '*v set*
⟨*proof*⟩

sublocale *renaming-variables* **where**
vars = *vars* :: ('*f*, '*v*) *term* ⇒ '*v set* **and** *subst* = (·*t*) **and** *id-subst* = *Var* **and**
comp-subst = (⊙)
⟨*proof*⟩

sublocale *term-grounding* **where**
subst = (·*t*) **and** *id-subst* = *Var* **and** *comp-subst* = (⊙) **and**
vars = *vars* :: ('*f*, '*v*) *term* ⇒ '*v set* **and** *from-ground* = *from-ground* **and**
to-ground = *to-ground*
⟨*proof*⟩

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

declare *Term-Context.ground-vars-term-empty* [*simp del*]

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

end

1.3 Setup for lifting from terms

locale *lifting* =
based-functional-substitution-lifting +
all-subst-ident-iff-ground-lifting +
grounding-lifting +
renaming-variables-lifting +
variables-in-base-imgu-lifting

locale *term-based-lifting* =
term: *nonground-term* +
lifting **where**
comp-subst = $(\odot)$ **and** *id-subst* = *Var* **and** *base-subst* = $(\cdot t)$ **and** *base-vars* = *term.vars*

end
theory *Nonground-Context*
imports
Nonground-Term
Ground-Context
begin

2 Nonground Contexts and Substitutions

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

abbreviation *subst-apply-ctxt* ::
$('f, 'v)$ *context* ⇒ $('f, 'v)$ *subst* ⇒ $('f, 'v)$ *context* (**infixl** $\cdot t_c$ *67*) **where**
subst-apply-ctxt ≡ *subst-apply-actxt*

global-interpretation *context*: *finite-natural-functor* **where**
map = *map-args-actxt* **and** *to-set* = *set2-actxt*
⟨*proof*⟩

global-interpretation *context*: *natural-functor-conversion* **where**
map = *map-args-actxt* **and** *to-set* = *set2-actxt* **and** *map-to* = *map-args-actxt*
and
map-from = *map-args-actxt* **and** *map′* = *map-args-actxt* **and** *to-set′* = *set2-actxt*
⟨*proof*⟩

locale *nonground-context* =
term: *nonground-term*
begin

sublocale *term-based-lifting* **where**
sub-subst = $(\cdot t)$ **and** *sub-vars* = *term.vars* **and**

to-set = *set2-actxt* :: ('*f*, '*v*) *context* ⇒ ('*f*, '*v*) *term set* **and** *map* = *map-args-actxt*
and
sub-to-ground = *term.to-ground* **and** *sub-from-ground* = *term.from-ground* **and**
to-ground-map = *map-args-actxt* **and** *from-ground-map* = *map-args-actxt* **and**
ground-map = *map-args-actxt* **and** *to-set-ground* = *set2-actxt*
rewrites
⋀*c* *σ*. *subst* *c* *σ* = *c* ·$_t_c$ *σ* **and**
⋀*c*. *vars* *c* = *vars-ctxt* *c*
⟨*proof*⟩

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

lemma *term-to-ground-context-to-ground* [*simp*]:
shows *term.to-ground* *c*⟨*t*⟩ = (*to-ground* *c*)⟨*term.to-ground* *t*⟩$_G$
⟨*proof*⟩

lemma *term-from-ground-context-from-ground* [*simp*]:
term.from-ground *c*$_G$⟨*t*$_G$⟩$_G$ = (*from-ground* *c*$_G$)⟨*term.from-ground* *t*$_G$⟩
⟨*proof*⟩

lemma *term-from-ground-context-to-ground*:
assumes *is-ground* *c*
shows *term.from-ground* (*to-ground* *c*)⟨*t*$_G$⟩$_G$ = *c*⟨*term.from-ground* *t*$_G$⟩
⟨*proof*⟩

lemmas *safe-unfolds* =
eval-ctxt
term-to-ground-context-to-ground
term-from-ground-context-from-ground

lemma *composed-context-is-ground* [*simp*]:
is-ground (*c* ∘$_c$ *c*′) ⟷ *is-ground* *c* ∧ *is-ground* *c*′
⟨*proof*⟩

lemma *ground-context-subst*:
assumes
is-ground *c*$_G$
c$_G$ = (*c* ·$_t_c$ *σ*) ∘$_c$ *c*′
shows
c$_G$ = *c* ∘$_c$ *c*′ ·$_t_c$ *σ*
⟨*proof*⟩

lemma *from-ground-hole* [*simp*]: *from-ground* *c*$_G$ = □ ⟷ *c*$_G$ = □
⟨*proof*⟩

lemma *hole-simps* [*simp*]: *from-ground* □ = □ *to-ground* □ = □
⟨*proof*⟩

lemma *term-with-context-is-ground* [*simp*]:
term.is-ground $c\langle t\rangle \longleftrightarrow$ *is-ground* $c \wedge$ *term.is-ground* t
⟨*proof*⟩

lemma *map-args-actxt-compose* [*simp*]:
map-args-actxt f $(c \circ_c c') =$ *map-args-actxt* f $c \circ_c$ *map-args-actxt* f c'
⟨*proof*⟩

lemma *from-ground-compose* [*simp*]: *from-ground* $(c \circ_c c') =$ *from-ground* $c \circ_c$ *from-ground* c'
⟨*proof*⟩

lemma *to-ground-compose* [*simp*]: *to-ground* $(c \circ_c c') =$ *to-ground* $c \circ_c$ *to-ground* c'
⟨*proof*⟩

end

locale *nonground-term-with-context* =
term: *nonground-term* +
context: *nonground-context*

end
theory *Multiset-Grounding-Lifting*
imports
HOL−Library.Multiset
Abstract-Substitution.Functional-Substitution-Lifting
begin

locale *multiset-grounding-lifting* =
functional-substitution-lifting **where** *to-set* = *set-mset* **and** *map* = *image-mset* +
grounding-lifting **where**
to-set = *set-mset* **and** *map* = *image-mset* **and** *to-ground-map* = *image-mset* **and**
from-ground-map = *image-mset* **and** *ground-map* = *image-mset* **and** *to-set-ground* = *set-mset*
begin

sublocale *natural-magma-with-empty-grounding-lifting* **where**
plus = (+) **and** *wrap* = $\lambda l.$ $\{\#l\#\}$ **and** *plus-ground* = (+) **and** *wrap-ground* = $\lambda l.$ $\{\#l\#\}$ **and**
empty = $\{\#\}$ **and** *empty-ground* = $\{\#\}$ **and** *to-set* = *set-mset* **and** *map* = *image-mset* **and**
to-ground-map = *image-mset* **and** *from-ground-map* = *image-mset* **and** *ground-map* = *image-mset* **and**
to-set-ground = *set-mset* **and** *add* = *add-mset* **and** *add-ground* = *add-mset*
⟨*proof*⟩

sublocale *natural-magma-functor-functional-substitution-lifting* **where**
plus = (+) **and** *wrap* = λ*l*. {#*l*#} **and** *to-set* = *set-mset* **and** *map* = *image-mset* **and** *add* = *add-mset*
⟨*proof*⟩

end

end
theory *Nonground-Clause*
imports
Ground-Clause
Nonground-Term
Nonground-Context
Clausal-Calculus-Extra
Multiset-Extra
Multiset-Grounding-Lifting
begin

3 Nonground Clauses and Substitutions

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

locale *term-based-multiset-lifting* =
term-based-lifting **where**
map = *image-mset* **and** *to-set* = *set-mset* **and** *to-ground-map* = *image-mset* **and** *from-ground-map* = *image-mset* **and** *ground-map* = *image-mset* **and** *to-set-ground* = *set-mset*
begin

sublocale *multiset-grounding-lifting* **where**
id-subst = *Var* **and** *comp-subst* = (⊙)
⟨*proof*⟩

end

locale *nonground-clause* = *nonground-term-with-context*
begin

3.1 Nonground Atoms

sublocale *atom*: *term-based-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*⟩

notation *atom.subst* (**infixl** $\cdot a$ *67*)

lemma *vars-atom* [*simp*]: *atom.vars* (*Upair* t_1 t_2) = *term.vars* t_1 ∪ *term.vars* t_2
⟨*proof*⟩

lemma *subst-atom* [*simp*]:
Upair t_1 t_2 $\cdot a$ σ = *Upair* (t_1 $\cdot t$ σ) (t_2 $\cdot t$ σ)
⟨*proof*⟩

lemma *atom-from-ground-term-from-ground* [*simp*]:
atom.from-ground (*Upair* t_{G1} t_{G2}) = *Upair* (*term.from-ground* t_{G1}) (*term.from-ground* t_{G2})
⟨*proof*⟩

lemma *atom-to-ground-term-to-ground* [*simp*]:
atom.to-ground (*Upair* t_1 t_2) = *Upair* (*term.to-ground* t_1) (*term.to-ground* t_2)
⟨*proof*⟩

lemma *atom-is-ground-term-is-ground* [*simp*]:
atom.is-ground (*Upair* t_1 t_2) ⟷ *term.is-ground* t_1 ∧ *term.is-ground* t_2
⟨*proof*⟩

lemma *obtain-from-atom-subst*:
assumes *Upair* $t_1{}'$ $t_2{}'$ = a $\cdot a$ σ
obtains t_1 t_2
where a = *Upair* t_1 t_2 $t_1{}'$ = t_1 $\cdot t$ σ $t_2{}'$ = t_2 $\cdot t$ σ
⟨*proof*⟩

3.2 Nonground Literals

sublocale *literal*: *term-based-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*⟩

notation *literal.subst* (**infixl** $\cdot l$ *66*)

lemma *vars-literal* [*simp*]:
literal.vars (*Pos* a) = *atom.vars* a
literal.vars (*Neg* a) = *atom.vars* a
literal.vars ((*if* b *then Pos else Neg*) a) = *atom.vars* a
⟨*proof*⟩

lemma *subst-literal* [*simp*]:
Pos a $\cdot l$ σ = *Pos* (a $\cdot a$ σ)

Neg a ·l σ = Neg (a ·a σ)
atm-of (l ·l σ) = atm-of l ·a σ
⟨*proof*⟩

lemma *subst-literal-if* [*simp*]:
(if b then Pos else Neg) a ·l ϱ = (if b then Pos else Neg) (a ·a ϱ)
⟨*proof*⟩

lemma *subst-polarity-stable*:
shows
subst-neg-stable [*simp*]: *is-neg (l ·l σ) ⟷ is-neg l* **and**
subst-pos-stable [*simp*]: *is-pos (l ·l σ) ⟷ is-pos l*
⟨*proof*⟩

declare *literal.discI* [*intro*]

lemma *literal-from-ground-atom-from-ground* [*simp*]:
literal.from-ground (Neg a_G) = Neg (atom.from-ground a_G)
literal.from-ground (Pos a_G) = Pos (atom.from-ground a_G)
⟨*proof*⟩

lemma *literal-from-ground-polarity-stable* [*simp*]:
shows
neg-literal-from-ground-stable: *is-neg (literal.from-ground l_G) ⟷ is-neg l_G* **and**
pos-literal-from-ground-stable: *is-pos (literal.from-ground l_G) ⟷ is-pos l_G*
⟨*proof*⟩

lemma *literal-to-ground-atom-to-ground* [*simp*]:
literal.to-ground (Pos a) = Pos (atom.to-ground a)
literal.to-ground (Neg a) = Neg (atom.to-ground a)
⟨*proof*⟩

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

lemma *obtain-from-pos-literal-subst*:
assumes *l ·l σ = t_1′ ≈ t_2′*
obtains *t_1 t_2*
where *l = t_1 ≈ t_2 t_1′ = t_1 ·t σ t_2′ = t_2 ·t σ*
⟨*proof*⟩

lemma *obtain-from-neg-literal-subst*:
assumes *l ·l σ = t_1′ !≈ t_2′*
obtains *t_1 t_2*
where *l = t_1 !≈ t_2 t_1 ·t σ = t_1′ t_2 ·t σ = t_2′*
⟨*proof*⟩

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

3.3 Nonground Literals - Alternative

lemma *uprod-literal-subst-eq-literal-subst*: *map-uprod-literal* $(\lambda t.\ t \cdot t\ \sigma)\ l = l \cdot l\ \sigma$
⟨*proof*⟩

lemma *uprod-literal-vars-eq-literal-vars*: $\bigcup$ (*term.vars* ‘ *uprod-literal-to-set l*) = *literal.vars l*
⟨*proof*⟩

lemma *uprod-literal-from-ground-eq-literal-from-ground*:
map-uprod-literal term.from-ground l_G = *literal.from-ground* l_G
⟨*proof*⟩

lemma *uprod-literal-to-ground-eq-literal-to-ground*:
map-uprod-literal term.to-ground l = *literal.to-ground l*
⟨*proof*⟩

sublocale *uprod-literal*: *term-based-lifting* **where**
sub-subst = $(\cdot t)$ **and** *sub-vars* = *term.vars* **and** *map* = *map-uprod-literal* **and**
to-set = *uprod-literal-to-set* **and** *sub-to-ground* = *term.to-ground* **and**
sub-from-ground = *term.from-ground* **and** *to-ground-map* = *map-uprod-literal* **and**
from-ground-map = *map-uprod-literal* **and** *ground-map* = *map-uprod-literal* **and**
to-set-ground = *uprod-literal-to-set*
rewrites
uprod-literal-subst [*simp*]: $\bigwedge l\ \sigma$. *uprod-literal.subst l* σ = *literal.subst l* σ **and**
uprod-literal-vars [*simp*]: $\bigwedge l$. *uprod-literal.vars l* = *literal.vars l* **and**
uprod-literal-from-ground [*simp*]: $\bigwedge l_G$. *uprod-literal.from-ground* l_G = *literal.from-ground* l_G **and**
uprod-literal-to-ground [*simp*]:$\bigwedge l$. *uprod-literal.to-ground l* = *literal.to-ground l*
⟨*proof*⟩

lemma *mset-literal-from-ground*:
mset-lit (*literal.from-ground l*) = *image-mset term.from-ground* (*mset-lit l*)
⟨*proof*⟩

3.4 Nonground Clauses

sublocale *clause*: *term-based-multiset-lifting* **where**
sub-subst = *literal.subst* **and** *sub-vars* = *literal.vars* **and** *sub-to-ground* = *literal.to-ground* **and**
sub-from-ground = *literal.from-ground*
⟨*proof*⟩

notation *clause.subst* (**infixl** $\cdot$ *67*)

lemmas *clause-submset-vars-clause-subset* [*intro*] =
clause.to-set-subset-vars-subset[*OF set-mset-mono*]

lemmas *sub-ground-clause* = *clause.to-set-subset-is-ground*[*OF set-mset-mono*]

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

lemma *clause-from-ground-remove1-mset* [*simp*]:
clause.from-ground (*remove1-mset* $l_G\ C_G$) =
remove1-mset (*literal.from-ground* l_G) (*clause.from-ground* C_G)
⟨*proof*⟩

lemmas *clause-safe-unfolds* =
atom-to-ground-term-to-ground
literal-to-ground-atom-to-ground
atom-from-ground-term-from-ground
literal-from-ground-atom-from-ground
literal-from-ground-polarity-stable
subst-atom
subst-literal
vars-atom
vars-literal

end

end
theory *Selection-Function*
imports *Ordered-Resolution-Prover.Clausal-Logic*
begin

locale *selection-function* =
fixes *select* :: *'a clause* ⇒ *'a clause*
assumes
select-subset: $\bigwedge C.$ *select* $C \subseteq\# C$ **and**
select-negative-literals: $\bigwedge C\ l.\ l \in\#$ *select* $C \Longrightarrow$ *is-neg* l

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

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

context *nonground-clause*
begin

definition *is-select-grounding* :: $('f, 'v)$ *select* ⇒ *'f ground-select* ⇒ *bool* **where**

is-select-grounding select select$_G \equiv \forall\, C_G.\ \exists\, C\ \gamma.$
clause.is-ground $(C \cdot \gamma) \wedge$
$C_G =$ *clause.to-ground* $(C \cdot \gamma) \wedge$
select$_G$ $C_G =$ *clause.to-ground* $((\mathit{select}\ C) \cdot \gamma)$

end

locale *nonground-selection-function* =
nonground-clause +
selection-function select
for *select* :: $('f, 'v)$ *atom clause* $\Rightarrow$ $('f, 'v)$ *atom clause*
begin

abbreviation *is-grounding* :: $'f$ *ground-select* $\Rightarrow$ *bool* **where**
is-grounding select$_G \equiv$ *is-select-grounding select select*$_G$

definition *select*$_{Gs}$ **where**
select$_{Gs} = \{$ *select*$_G$. *is-grounding select*$_G$ $\}$

definition *select*$_G$*-simple* **where**
select$_G$*-simple* $C =$ *clause.to-ground* (*select* (*clause.from-ground* C))

lemma *select*$_G$*-simple*: *is-grounding select*$_G$*-simple*
⟨*proof*⟩

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

lemma *is-ground-in-selection*:
assumes $l \in\#$ *select* (*clause.from-ground* C)
shows *literal.is-ground* l
⟨*proof*⟩

lemma *ground-literal-in-selection*:
assumes *clause.is-ground* C $l_G \in\#$ *clause.to-ground* C
shows *literal.from-ground* $l_G \in\#$ C
⟨*proof*⟩

lemma *select-ground-subst*:
assumes *clause.is-ground* $(C \cdot \gamma)$
shows *clause.is-ground* (*select* $C \cdot \gamma$)
⟨*proof*⟩

lemma *select-neg-subst*:
assumes $l \in\#$ *select* $C \cdot \gamma$
shows *is-neg* l
⟨*proof*⟩

lemma *select-vars-subset*: $\bigwedge C.$ *clause.vars* (*select* C) $\subseteq$ *clause.vars* C
⟨*proof*⟩

end

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

lemma *Collect-if-eq*: $\{x.$ *if* b x *then* P x *else* Q x $\} = \{x.$ b $x \wedge P$ x $\} \cup \{x.$ $\neg b$ $x \wedge Q$ $x\}$
⟨*proof*⟩

lemma *Collect-not-mem-conj-eq*: $\{x.$ $x \notin X \wedge P$ $x\} = \{x.$ P $x\} - X$
⟨*proof*⟩

end
theory *Infinite-Variables-Per-Type*
imports
HOL−Library.Countable-Set
HOL−Cardinals.Cardinals
Fresh-Identifiers.Fresh
Collect-Extra
begin

lemma *infinite-prods*:
assumes *infinite* (*UNIV* :: *'a set*)
shows *infinite* $\{p$ :: *'a* $\times$ *'a. fst* $p = x\}$
⟨*proof*⟩

lemma *surj-infinite-set*: *surj* $g \Longrightarrow$ *infinite* $\{x.$ f $x = \tau\} \Longrightarrow$ *infinite* $\{x.$ f $(g$ $x) = \tau\}$
⟨*proof*⟩

definition *infinite-variables-per-type-on* :: *'var set* $\Rightarrow$ (*'var* $\Rightarrow$ *'ty*) $\Rightarrow$ *bool* **where**
infinite-variables-per-type-on X $\mathcal{V} \equiv \forall \tau \in \mathcal{V}$ ‘ $X.$ *infinite* $\{x.$ $\mathcal{V}$ $x = \tau\}$

abbreviation *infinite-variables-per-type* :: (*'var* $\Rightarrow$ *'ty*) $\Rightarrow$ *bool* **where**
infinite-variables-per-type $\equiv$ *infinite-variables-per-type-on UNIV*

lemma *obtain-type-preserving-inj*:
fixes $\mathcal{V}$ :: *'v* $\Rightarrow$ *'ty*
assumes
finite-X: *finite* X **and**
$\mathcal{V}$: *infinite-variables-per-type* $\mathcal{V}$
obtains f :: *'v* $\Rightarrow$ *'v* **where**
inj f

$X \cap f \mathbin{`} Y = \{\}$
$\forall x \in Y.\ \mathcal{V}\ (f\ x) = \mathcal{V}\ x$
⟨*proof*⟩

lemma *obtain-type-preserving-injs*:
fixes $\mathcal{V}_1\ \mathcal{V}_2 :: 'v \Rightarrow 'ty$
assumes
finite-X: *finite* X **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$
obtains $f\ f' :: 'v \Rightarrow 'v$ **where**
inj f inj f'
$f \mathbin{`} X \cap f' \mathbin{`} Y = \{\}$
$\forall x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
$\forall x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
⟨*proof*⟩

lemma *obtain-type-preserving-injs'*:
fixes $\mathcal{V}_1\ \mathcal{V}_2 :: 'v \Rightarrow 'ty$
assumes
finite-Y: *finite* Y **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$
obtains $f\ f' :: 'v \Rightarrow 'v$ **where**
inj f inj f'
$f \mathbin{`} X \cap f' \mathbin{`} Y = \{\}$
$\forall x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
$\forall x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
⟨*proof*⟩

lemma *obtain-infinite-variables-per-type-on*:
assumes
infinite-UNIV: *infinite* (*UNIV* :: *'v set*) **and**
finite-Y: *finite* Y **and**
finite-Z: *finite* Z **and**
disjoint: $Y \cap Z = \{\}$
obtains $\mathcal{V} :: 'v \Rightarrow 'ty$
where *infinite-variables-per-type-on* $X\ \mathcal{V}\ \forall x \in Y.\ \mathcal{V}\ x = \mathcal{V}'\ x\ \forall x \in Z.\ \mathcal{V}\ x = \mathcal{V}''\ x$
⟨*proof*⟩

lemma *obtain-infinite-variables-per-type-on'*:
assumes *infinite-UNIV*: *infinite* (*UNIV* :: *'v set*) **and** *finite-Y*: *finite* Y
obtains $\mathcal{V} :: 'v \Rightarrow 'ty$
where *infinite-variables-per-type-on* $X\ \mathcal{V}\ \forall x \in Y.\ \mathcal{V}\ x = \mathcal{V}'\ x$
⟨*proof*⟩

lemma *obtain-infinite-variables-per-type-on''*:
assumes *finite* Y
obtains $\mathcal{V} :: 'v :: infinite \Rightarrow 'ty$
where *infinite-variables-per-type-on* $X\ \mathcal{V}\ \forall x \in Y.\ \mathcal{V}\ x = \mathcal{V}'\ x$

⟨*proof*⟩

lemma *infinite-variables-per-type-on-subset*:
$X \subseteq Y \Longrightarrow$ *infinite-variables-per-type-on* $Y\ \mathcal{V} \Longrightarrow$ *infinite-variables-per-type-on* $X\ \mathcal{V}$
⟨*proof*⟩

definition *infinite-variables-for-all-types* :: $('v \Rightarrow 'ty) \Rightarrow$ *bool* **where**
infinite-variables-for-all-types $\mathcal{V} \equiv \forall \tau.$ *infinite* $\{x.\ \mathcal{V}\ x = \tau\}$

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

lemma *obtain-infinite-variables-for-all-types*:
assumes $|UNIV :: 'ty\ set| \leq_o |UNIV :: 'v\ set|$
obtains $\mathcal{V} :: 'v :: infinite \Rightarrow 'ty$ **where** *infinite-variables-for-all-types* $\mathcal{V}$
⟨*proof*⟩

lemma *infinite-variables-per-type-if-infinite-variables-for-all-types*:
infinite-variables-for-all-types $\mathcal{V} \Longrightarrow$ *infinite-variables-per-type* $\mathcal{V}$
⟨*proof*⟩

end
theory *Typed-Functional-Substitution*
imports
Typing
Abstract-Substitution.Functional-Substitution
Infinite-Variables-Per-Type
begin

type-synonym $('v, 'ty)$ *var-types* $= 'v \Rightarrow 'ty$

locale *base-typed-functional-substitution* =
base-functional-substitution **where** *id-subst* = *id-subst*
for
id-subst :: $'v \Rightarrow 'base$ **and**
welltyped :: $('v, 'ty)$ *var-types* $\Rightarrow$ *'base* $\Rightarrow$ *'ty* $\Rightarrow$ *bool* +
assumes
base-typing: $\bigwedge \mathcal{V}.$ *Typing.typing* (*welltyped* $\mathcal{V}$) **and**
typed-id-subst [*intro*]: $\bigwedge \mathcal{V}\ x.$ *welltyped* $\mathcal{V}$ (*id-subst* x) ($\mathcal{V}\ x$)
begin

sublocale *typing welltyped* $\mathcal{V}$
⟨*proof*⟩

abbreviation *is-welltyped-on* :: $'v\ set \Rightarrow ('v, 'ty)$ *var-types* $\Rightarrow ('v \Rightarrow 'base) \Rightarrow$ *bool*
where

is-welltyped-on $X\ \mathcal{V}\ \sigma \equiv \forall\, x \in X.\ \mathit{welltyped}\ \mathcal{V}\ (\sigma\ x)\ (\mathcal{V}\ x)$

lemma *subst-update*:
assumes *welltyped* $\mathcal{V}$ (*id-subst var*) τ *welltyped* $\mathcal{V}$ *update* τ *is-welltyped-on* $X\ \mathcal{V}$ γ
shows *is-welltyped-on* $X\ \mathcal{V}$ (γ(*var* := *update*))
⟨*proof*⟩

lemma *is-typed-on-subset*:
assumes *is-welltyped-on* $Y\ \mathcal{V}\ \sigma\ X \subseteq Y$
shows *is-welltyped-on* $X\ \mathcal{V}\ \sigma$
⟨*proof*⟩

lemma *is-typed-id-subst* [*intro*]: *is-welltyped-on* $X\ \mathcal{V}$ *id-subst*
⟨*proof*⟩

end

locale *typed-functional-substitution* =
base: *base-typed-functional-substitution* **where**
subst = *base-subst* **and** *vars* = *base-vars* **and** *welltyped* = *base-welltyped* +
based-functional-substitution **where** *vars* = *vars*
for
base-welltyped :: ('*v*, '*ty*) *var-types* ⇒ '*base* ⇒ '*ty* ⇒ *bool* **and**
vars :: '*expr* ⇒ '*v set* **and**
welltyped :: ('*v*, '*ty*) *var-types* ⇒ '*expr* ⇒ '*ty*' ⇒ *bool* +
assumes
base-typing0: $\bigwedge\mathcal{V}$. *Typing.typing* (*welltyped* $\mathcal{V}$)
begin

sublocale *typing welltyped* $\mathcal{V}$
⟨*proof*⟩

abbreviation *is-welltyped-ground-instance* **where**
is-welltyped-ground-instance expr $\mathcal{V}\ \gamma \equiv$
is-ground (*expr* · γ) ∧
(∃ τ. *welltyped* $\mathcal{V}$ *expr* τ) ∧
base.is-welltyped-on (*vars expr*) $\mathcal{V}\ \gamma$ ∧
infinite-variables-per-type $\mathcal{V}$

end

locale *inhabited-typed-functional-substitution* =
typed-functional-substitution +
assumes *types-inhabited*: $\bigwedge\mathcal{V}\ \tau$. ∃ *b*. *base.is-ground b* ∧ *base-welltyped* $\mathcal{V}$ *b* τ
begin

lemma *ground-subst-extension*:
assumes
grounding: *is-ground* (*expr* · γ) **and**
γ-*is-typed-on*: *base.is-welltyped-on* (*vars expr*) $\mathcal{V}$ γ
obtains γ'
where
base.is-ground-subst γ'
base.is-welltyped-on UNIV $\mathcal{V}$ γ'
$\forall x \in$ *vars expr*. γ $x = \gamma'$ x
⟨*proof*⟩

lemma *grounding-extension*:
assumes
grounding: *is-ground* (*expr* · γ) **and**
γ-*is-typed-on*: *base.is-welltyped-on* (*vars expr*) $\mathcal{V}$ γ
obtains γ'
where
is-ground (*expr'* · γ')
base.is-welltyped-on (*vars expr'*) $\mathcal{V}$ γ'
$\forall x \in$ *vars expr*. γ $x = \gamma'$ x
⟨*proof*⟩

end

sublocale *base-typed-functional-substitution* ⊆ *typed-functional-substitution* **where**
base-subst = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped*
⟨*proof*⟩

locale *typed-grounding-functional-substitution* =
typed-functional-substitution + *grounding*
begin

definition *welltyped-ground-instances* **where**
welltyped-ground-instances typed-expr =
{ *to-ground* (*fst typed-expr* · γ) | γ.
is-welltyped-ground-instance (*fst typed-expr*) (*snd typed-expr*) γ }

lemma *typed-ground-instances-ground-instances'*:
welltyped-ground-instances (*expr*, $\mathcal{V}$) ⊆ *ground-instances'* *expr*
⟨*proof*⟩

end

locale *typed-subst-stability* = *typed-functional-substitution* +
assumes
subst-stability [*simp*]: $\bigwedge \mathcal{V}$ *expr* σ τ.
base.is-welltyped-on (*vars expr*) $\mathcal{V}$ σ ⟹ *welltyped* $\mathcal{V}$ (*expr* · σ) τ ⟷ *welltyped* $\mathcal{V}$ *expr* τ

begin

lemma *subst-stability-UNIV* [*simp*]:
base.is-welltyped-on UNIV $\mathcal{V}$ $\sigma \Longrightarrow$ *welltyped* $\mathcal{V}$ (*expr* $\cdot$ σ) $\tau \longleftrightarrow$ *welltyped* $\mathcal{V}$ *expr* τ
⟨*proof*⟩

end

locale *base-typed-subst-stability* =
base-typed-functional-substitution +
typed-subst-stability **where** *base-subst* = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped*
begin

lemma *typed-subst-compose* [*intro*]:
assumes
is-welltyped-on X $\mathcal{V}$ σ
is-welltyped-on ($\bigcup$(*vars* ‘ σ ‘ *X*)) $\mathcal{V}$ σ'
shows *is-welltyped-on X* $\mathcal{V}$ ($\sigma \odot \sigma'$)
⟨*proof*⟩

lemma *typed-subst-compose-UNIV* [*intro*]:
assumes
is-welltyped-on UNIV $\mathcal{V}$ σ
is-welltyped-on UNIV $\mathcal{V}$ σ'
shows *is-welltyped-on UNIV* $\mathcal{V}$ ($\sigma \odot \sigma'$)
⟨*proof*⟩

end

locale *replaceable-*$\mathcal{V}$ = *typed-functional-substitution* +
assumes *replace-*$\mathcal{V}$:
$\bigwedge$*expr* $\mathcal{V}$ $\mathcal{V}'$ τ. $\forall$ *x*$\in$ *vars expr*. $\mathcal{V}$ *x* = $\mathcal{V}'$ *x* $\Longrightarrow$ *welltyped* $\mathcal{V}$ *expr* $\tau \Longrightarrow$ *welltyped* $\mathcal{V}'$ *expr* τ
begin

lemma *replace-*$\mathcal{V}$*-iff*:
assumes $\forall$ *x*$\in$ *vars expr*. $\mathcal{V}$ *x* = $\mathcal{V}'$ *x*
shows *welltyped* $\mathcal{V}$ *expr* $\tau \longleftrightarrow$ *welltyped* $\mathcal{V}'$ *expr* τ
⟨*proof*⟩

lemma *is-ground-typed*:
assumes *is-ground expr*
shows *welltyped* $\mathcal{V}$ *expr* $\tau \longleftrightarrow$ *welltyped* $\mathcal{V}'$ *expr* τ
⟨*proof*⟩

end

locale *typed-renaming* =
  typed-functional-substitution +
  renaming-variables +
  assumes
    typed-renaming [*simp*]:
    $\bigwedge \mathcal{V}\ \mathcal{V}'\ expr\ \varrho\ \tau.\ base.is\text{-}renaming\ \varrho \Longrightarrow$
      $\forall\, x \in vars\ expr.\ \mathcal{V}\ x = \mathcal{V}'\ (rename\ \varrho\ x) \Longrightarrow$
      $welltyped\ \mathcal{V}'\ (expr \cdot \varrho)\ \tau \longleftrightarrow welltyped\ \mathcal{V}\ expr\ \tau$

locale *base-typed-renaming* =
  base-typed-functional-substitution **where**
    welltyped = *welltyped* +
  typed-renaming **where**
    base-subst = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped* **and**
    welltyped = *welltyped* +
  replaceable-$\mathcal{V}$ **where**
    base-subst = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped* **and**
    welltyped = *welltyped*
  for *welltyped* :: ('*v*, '*ty*) *var-types* ⇒ '*expr* ⇒ '*ty* ⇒ *bool*
begin

lemma *renaming-ground-subst*:
  assumes
    is-renaming ϱ
    is-welltyped-on $(\bigcup (vars\ `\ \varrho\ `\ X))\ \mathcal{V}'\ \gamma$
    is-welltyped-on $X\ \mathcal{V}\ \varrho$
    is-ground-subst γ
    $\forall\, x \in X.\ \mathcal{V}\ x = \mathcal{V}'\ (rename\ \varrho\ x)$
  shows *is-welltyped-on* $X\ \mathcal{V}\ (\varrho \odot \gamma)$
⟨*proof*⟩

lemma *obtain-typed-renaming*:
  fixes $\mathcal{V}$ :: '*v* ⇒ '*ty*
  assumes
    finite X
    infinite-variables-per-type $\mathcal{V}$
  obtains ϱ :: '*v* ⇒ '*expr* **where**
    is-renaming ϱ
    id-subst $` X \cap \varrho\ ` Y = \{\}$
    is-welltyped-on $Y\ \mathcal{V}\ \varrho$
⟨*proof*⟩

lemma *obtain-typed-renamings*:
  fixes $\mathcal{V}_1\ \mathcal{V}_2$ :: '*v* ⇒ '*ty*
  assumes
    finite X
    infinite-variables-per-type $\mathcal{V}_2$
  obtains $\varrho_1\ \varrho_2$ :: '*v* ⇒ '*expr* **where**
    is-renaming ϱ_1

is-renaming ϱ_2
ϱ_1 ‘ $X \cap \varrho_2$ ‘ $Y = \{\}$
is-welltyped-on X $\mathcal{V}_1$ ϱ_1
is-welltyped-on Y $\mathcal{V}_2$ ϱ_2
⟨*proof*⟩

lemma *obtain-typed-renamings'*:
fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: $'v \Rightarrow 'ty$
assumes
finite Y
infinite-variables-per-type $\mathcal{V}_1$
obtains ϱ_1 ϱ_2 :: $'v \Rightarrow 'expr$ **where**
is-renaming ϱ_1
is-renaming ϱ_2
ϱ_1 ‘ $X \cap \varrho_2$ ‘ $Y = \{\}$
is-welltyped-on X $\mathcal{V}_1$ ϱ_1
is-welltyped-on Y $\mathcal{V}_2$ ϱ_2
⟨*proof*⟩

lemma *renaming-subst-compose*:
assumes
is-renaming ϱ
is-welltyped-on X $\mathcal{V}$ $(\varrho \odot \sigma)$
is-welltyped-on X $\mathcal{V}$ ϱ
shows *is-welltyped-on* $(\bigcup (vars$ ‘ ϱ ‘ $X))$ $\mathcal{V}$ σ
⟨*proof*⟩

end

lemma (**in** *renaming-variables*) *obtain-merged-*$\mathcal{V}$:
assumes
ϱ_1: *is-renaming* ϱ_1 **and**
ϱ_2: *is-renaming* ϱ_2 **and**
rename-apart: *vars* $(expr \cdot \varrho_1) \cap vars\ (expr' \cdot \varrho_2) = \{\}$ **and**
finite-vars: *finite* $(vars\ expr)$ *finite* $(vars\ expr')$ **and**
infinite-UNIV: *infinite* $(UNIV :: 'a\ set)$
obtains $\mathcal{V}_3$ **where**
infinite-variables-per-type-on X $\mathcal{V}_3$
$\forall x{\in}vars\ expr.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (rename\ \varrho_1\ x)$
$\forall x{\in}vars\ expr'.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (rename\ \varrho_2\ x)$
⟨*proof*⟩

lemma (**in** *renaming-variables*) *obtain-merged-*$\mathcal{V}$*-infinite-variables-for-all-types*:
assumes
ϱ_1: *is-renaming* ϱ_1 **and**
ϱ_2: *is-renaming* ϱ_2 **and**
rename-apart: *vars* $(expr \cdot \varrho_1) \cap vars\ (expr' \cdot \varrho_2) = \{\}$ **and**
$\mathcal{V}_2$: *infinite-variables-for-all-types* $\mathcal{V}_2$ **and**
finite-vars: *finite* $(vars\ expr)$

obtains $\mathcal{V}_3$ **where**
$\forall\, x{\in}vars\ expr.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (rename\ \varrho_1\ x)$
$\forall\, x{\in}vars\ expr'.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (rename\ \varrho_2\ x)$
infinite-variables-for-all-types $\mathcal{V}_3$
⟨*proof*⟩

lemma (**in** *renaming-variables*) *obtain-merged-*$\mathcal{V}'$*-infinite-variables-for-all-types*:
assumes
ϱ_1: *is-renaming* ϱ_1 **and**
ϱ_2: *is-renaming* ϱ_2 **and**
rename-apart: $vars\ (expr \cdot \varrho_1) \cap vars\ (expr' \cdot \varrho_2) = \{\}$ **and**
$\mathcal{V}_1$: *infinite-variables-for-all-types* $\mathcal{V}_1$ **and**
finite-vars: *finite* $(vars\ expr')$
obtains $\mathcal{V}_3$ **where**
$\forall\, x{\in}vars\ expr.\ \mathcal{V}_1\ x = \mathcal{V}_3\ (rename\ \varrho_1\ x)$
$\forall\, x{\in}vars\ expr'.\ \mathcal{V}_2\ x = \mathcal{V}_3\ (rename\ \varrho_2\ x)$
infinite-variables-for-all-types $\mathcal{V}_3$
⟨*proof*⟩

locale *based-typed-renaming* =
base: *base-typed-renaming* **where**
$subst = base\text{-}subst$ **and** $vars = base\text{-}vars :: 'base \Rightarrow 'v\ set$ **and**
$welltyped = base\text{-}welltyped :: ('v \Rightarrow 'ty) \Rightarrow 'base \Rightarrow 'ty \Rightarrow bool +$
typed-renaming
begin

lemma *renaming-grounding*:
assumes
renaming: *base.is-renaming* ϱ **and**
ϱ-γ-*is-welltyped*: *base.is-welltyped-on* $(vars\ expr)\ \mathcal{V}\ (\varrho \odot \gamma)$ **and**
grounding: *is-ground* $(expr \cdot \varrho \odot \gamma)$ **and**
$\mathcal{V}$-$\mathcal{V}'$: $\forall\, x \in vars\ expr.\ \mathcal{V}\ x = \mathcal{V}'\ (rename\ \varrho\ x)$
shows *base.is-welltyped-on* $(vars\ (expr \cdot \varrho))\ \mathcal{V}'\ \gamma$
⟨*proof*⟩

lemma *obtain-merged-grounding*:
fixes $\mathcal{V}_1\ \mathcal{V}_2 :: 'v \Rightarrow 'ty$
assumes
base.is-welltyped-on $(vars\ expr)\ \mathcal{V}_1\ \gamma_1$
base.is-welltyped-on $(vars\ expr')\ \mathcal{V}_2\ \gamma_2$
is-ground $(expr \cdot \gamma_1)$
is-ground $(expr' \cdot \gamma_2)$ **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**
finite-vars: *finite* $(vars\ expr)$
obtains $\varrho_1\ \varrho_2\ \gamma$ **where**
base.is-renaming ϱ_1
base.is-renaming ϱ_2
$vars\ (expr \cdot \varrho_1) \cap vars\ (expr' \cdot \varrho_2) = \{\}$
base.is-welltyped-on $(vars\ expr)\ \mathcal{V}_1\ \varrho_1$

base.is-welltyped-on (*vars expr′*) $\mathcal{V}_2$ ϱ_2
$\forall\, x \in$ *vars expr*. $\gamma_1\ x = (\varrho_1 \odot \gamma)\ x$
$\forall\, x \in$ *vars expr′*. $\gamma_2\ x = (\varrho_2 \odot \gamma)\ x$
⟨*proof*⟩

lemma *obtain-merged-grounding′*:
fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: *'v* ⇒ *'ty*
assumes
typed-γ_1: *base.is-welltyped-on* (*vars expr*) $\mathcal{V}_1$ γ_1 **and**
typed-γ_2: *base.is-welltyped-on* (*vars expr′*) $\mathcal{V}_2$ γ_2 **and**
expr-grounding: *is-ground* (*expr* · γ_1) **and**
expr′-grounding: *is-ground* (*expr′* · γ_2) **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
finite-vars: *finite* (*vars expr′*)
obtains ϱ_1 ϱ_2 γ **where**
base.is-renaming ϱ_1
base.is-renaming ϱ_2
vars (*expr* · ϱ_1) ∩ *vars* (*expr′* · ϱ_2) = {}
base.is-welltyped-on (*vars expr*) $\mathcal{V}_1$ ϱ_1
base.is-welltyped-on (*vars expr′*) $\mathcal{V}_2$ ϱ_2
$\forall\, x \in$ *vars expr*. $\gamma_1\ x = (\varrho_1 \odot \gamma)\ x$
$\forall\, x \in$ *vars expr′*. $\gamma_2\ x = (\varrho_2 \odot \gamma)\ x$
⟨*proof*⟩

end

sublocale *base-typed-renaming* ⊆
based-typed-renaming **where** *base-vars* = *vars* **and** *base-subst* = *subst* **and** *base-welltyped* = *welltyped*
⟨*proof*⟩

locale *functional-substitution-typing* =
welltyped: *base-typed-functional-substitution* **where** *welltyped* = *welltyped*
for
welltyped :: (*'var*, *'ty*) *var-types* ⇒ *'expr* ⇒ *'ty* ⇒ *bool*
begin

abbreviation *is-welltyped-on* **where**
is-welltyped-on ≡ *welltyped.is-welltyped-on*

abbreviation *is-welltyped* **where**
is-welltyped ≡ *is-welltyped-on UNIV*

lemmas *welltyped-id-subst* = *welltyped.typed-id-subst*

lemmas *is-welltyped-id-subst* = *welltyped.is-typed-id-subst*

lemmas *is-welltyped-on-subset* = *welltyped.is-typed-on-subset*

lemmas *is-welltyped-subst-update* = *welltyped.subst-update*

end

end
theory *Typed-Functional-Substitution-Lifting*
imports
Typed-Functional-Substitution
Abstract-Substitution.Functional-Substitution-Lifting
begin

lemma *ext-equiv*: $(\bigwedge x.\ f\ x \equiv g\ x) \Longrightarrow f \equiv g$
⟨*proof*⟩

locale *typed-functional-substitution-lifting* =
sub: *typed-functional-substitution* **where**
vars = *sub-vars* **and** *subst* = *sub-subst* **and** *welltyped* = *sub-welltyped* **and** *base-vars* = *base-vars* +
based-functional-substitution-lifting **where** *to-set* = *to-set* **and** *base-vars* = *base-vars*
for
sub-welltyped :: ('*v*, '*ty*) *var-types* ⇒ '*sub* ⇒ '*ty*' ⇒ *bool* **and**
to-set :: '*expr* ⇒ '*sub set* **and**
base-vars :: '*base* ⇒ '*v set*
begin

sublocale *typing-lifting* **where** *sub-welltyped* = *sub-welltyped* $\mathcal{V}$
⟨*proof*⟩

sublocale *typed-functional-substitution* **where**
vars = *vars* **and** *subst* = *subst* **and** *welltyped* = *welltyped*
⟨*proof*⟩

end

locale *typed-grounding-functional-substitution-lifting* =
typed-functional-substitution-lifting +
grounding-lifting
begin

sublocale *typed-grounding-functional-substitution* **where**
vars = *vars* **and** *subst* = *subst* **and** *welltyped* = *welltyped* **and** *to-ground* = *to-ground* **and**
from-ground = *from-ground*
⟨*proof*⟩

end

locale *inhabited-typed-functional-substitution-lifting* =
  typed-functional-substitution-lifting +
  sub: *inhabited-typed-functional-substitution* **where**
  vars = *sub-vars* **and** *subst* = *sub-subst* **and** *welltyped* = *sub-welltyped*
begin

sublocale *inhabited-typed-functional-substitution* **where**
  vars = *vars* **and** *subst* = *subst* **and** *welltyped* = *welltyped*
  ⟨*proof*⟩

end

locale *typed-subst-stability-lifting* =
  typed-functional-substitution-lifting +
  sub: *typed-subst-stability* **where**
    welltyped = *sub-welltyped* **and** *vars* = *sub-vars* **and** *subst* = *sub-subst*
begin

sublocale *typed-subst-stability* **where** *welltyped* = *welltyped* **and** *subst* = *subst* **and** *vars* = *vars*
  ⟨*proof*⟩

end

locale *replaceable-$\mathcal{V}$-lifting* =
  typed-functional-substitution-lifting +
  sub: *replaceable-$\mathcal{V}$* **where** *welltyped* = *sub-welltyped* **and** *vars* = *sub-vars* **and** *subst* = *sub-subst*
begin

sublocale *replaceable-$\mathcal{V}$* **where**
  subst = *subst* **and** *vars* = *vars* **and** *welltyped* = *welltyped*
  ⟨*proof*⟩

end

locale *typed-renaming-lifting* =
  typed-functional-substitution-lifting **where**
    base-welltyped = *base-welltyped* :: (*'v* ⇒ *'ty*) ⇒ *'base* ⇒ *'ty* ⇒ *bool* +
  renaming-variables-lifting +
  sub: *based-typed-renaming* **where**
    subst = *sub-subst* **and** *vars* = *sub-vars* **and** *welltyped* = *sub-welltyped*
begin

sublocale *based-typed-renaming* **where**
  subst = *subst* **and** *vars* = *vars* **and** *welltyped* = *welltyped*
  ⟨*proof*⟩

end

end
theory *Nonground-Term-Typing*
imports
Term-Typing
Typed-Functional-Substitution
Nonground-Term
begin

locale *base-typed-properties* =
base-typed-renaming +
base-typed-subst-stability +
replaceable-$\mathcal{V}$ **where**
base-subst = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped* +
typed-grounding-functional-substitution **where**
base-subst = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped*

locale *base-typing-properties* =
welltyped: *base-typed-properties* **where** *welltyped* = *welltyped*
for *welltyped* :: (*'var*, *'ty*) *var-types* ⇒ *'base* ⇒ *'ty* ⇒ *bool*

locale *base-inhabited-typing-properties* =
base-typing-properties +
welltyped: *inhabited-typed-functional-substitution* **where**
welltyped = *welltyped* **and** *base-subst* = *subst* **and** *base-vars* = *vars* **and** *base-welltyped* = *welltyped*

locale *nonground-term-typing* =
term: *nonground-term* +
fixes $\mathcal{F}$ :: (*'f*, *'ty*) *fun-types*
begin

inductive *welltyped* :: (*'v*, *'ty*) *var-types* ⇒ (*'f*,*'v*) *term* ⇒ *'ty* ⇒ *bool*
for $\mathcal{V}$ **where**
Var: $\mathcal{V}$ *x* = *τ* ⟹ *welltyped* $\mathcal{V}$ (*Var x*) *τ*
| *Fun*: $\mathcal{F}$ *f* (*length ts*) = (*τs*, *τ*) ⟹ *list-all2* (*welltyped* $\mathcal{V}$) *ts τs* ⟹ *welltyped* $\mathcal{V}$ (*Fun f ts*) *τ*

sublocale *term*: *base-typing* **where**
welltyped = *welltyped* $\mathcal{V}$ **for** $\mathcal{V}$
⟨*proof*⟩

sublocale *term*: *term-typing* **where**
welltyped = *welltyped* $\mathcal{V}$ **and** *Fun* = *Fun* **for** $\mathcal{V}$
⟨*proof*⟩

sublocale *term*: *base-typing-properties* **where**
id-subst = *Var* :: *'v* ⇒ (*'f*, *'v*) *term* **and** *comp-subst* = (⊙) **and** *subst* = (·*t*) **and**
vars = *term.vars* **and** *welltyped* = *welltyped* **and** *to-ground* = *term.to-ground*
and
from-ground = *term.from-ground*
⟨*proof*⟩

sublocale *functional-substitution-typing* **where**
id-subst = *Var* :: *'v* ⇒ (*'f*, *'v*) *term* **and** *comp-subst* = (⊙) **and** *subst* = (·*t*) **and**
vars = *term.vars* **and** *welltyped* = *welltyped*
⟨*proof*⟩

end

locale *nonground-term-inhabited-typing* =
nonground-term-typing **where** $\mathcal{F} = \mathcal{F}$ **for** $\mathcal{F}$ :: (*'f*, *'ty*) *fun-types* +
assumes *types-inhabited*: $\bigwedge\tau.\ \exists f.\ \mathcal{F}\ f\ 0 = ([], \tau)$
begin

sublocale *base-inhabited-typing-properties* **where**
id-subst = *Var* :: *'v* ⇒ (*'f*, *'v*) *term* **and** *comp-subst* = (⊙) **and** *subst* = (·*t*) **and**
vars = *term.vars* **and** *welltyped* = *welltyped* **and** *to-ground* = *term.to-ground*
and
from-ground = *term.from-ground*
⟨*proof*⟩

end

end
theory *Nonground-Typing*
imports
Clause-Typing
Typed-Functional-Substitution-Lifting
Nonground-Term-Typing
Nonground-Clause
begin

type-synonym (*'f*, *'v*, *'ty*) *typed-clause* = (*'f*, *'v*) *atom clause* × (*'v*, *'ty*) *var-types*

locale *nonground-typed-lifting* =
typed-subst-stability-lifting +
replaceable-$\mathcal{V}$*-lifting* +
typed-renaming-lifting +
typed-grounding-functional-substitution-lifting

locale *nonground-typing-lifting* =
is-welltyped: *nonground-typed-lifting* **where**
sub-welltyped = *sub-welltyped* **and** *base-welltyped* = *base-welltyped*
for *base-welltyped sub-welltyped*

begin

abbreviation *is-welltyped-ground-instance* ≡ *is-welltyped.is-welltyped-ground-instance*

abbreviation *welltyped-ground-instances* ≡ *is-welltyped.welltyped-ground-instances*

lemmas *welltyped-ground-instances-def* = *is-welltyped.welltyped-ground-instances-def*

end

locale *nonground-inhabited-typing-lifting* =
nonground-typing-lifting +
welltyped: *inhabited-typed-functional-substitution-lifting* **where**
sub-welltyped = *sub-welltyped* **and** *base-welltyped* = *base-welltyped*

locale *term-based-nonground-typing-lifting* =
term: *nonground-term* +
nonground-typing-lifting **where**
id-subst = *Var* **and** *comp-subst* = (⊙) **and** *base-subst* = (·*t*) **and** *base-vars* = *term.vars*

locale *term-based-nonground-inhabited-typing-lifting* =
term: *nonground-term* +
nonground-inhabited-typing-lifting **where**
id-subst = *Var* **and** *comp-subst* = (⊙) **and** *base-subst* = (·*t*) **and** *base-vars* = *term.vars*

locale *nonground-typing* =
nonground-clause +
nonground-term-typing $\mathcal{F}$
for $\mathcal{F}$ :: ('*f*, '*ty*) *fun-types*
begin

sublocale *clause-typing welltyped* $\mathcal{V}$
⟨*proof*⟩

sublocale *atom*: *term-based-nonground-typing-lifting* **where**
base-welltyped = *welltyped* **and** *map* = *map-uprod* **and** *to-set* = *set-uprod* **and**
sub-welltyped = *welltyped* **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* **and**
sub-subst = (·*t*) **and** *sub-vars* = *term.vars*
⟨*proof*⟩

sublocale *literal*: *term-based-nonground-typing-lifting* **where**
base-welltyped = *welltyped* **and** *sub-vars* = *atom.vars* **and** *sub-subst* = (·*a*) **and**
map = *map-literal* **and** *to-set* = *set-literal* **and** *sub-welltyped* = *atom.welltyped* **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*⟩

sublocale *clause*: *term-based-nonground-typing-lifting* **where**
base-welltyped = *welltyped* **and** *sub-vars* = *literal.vars* **and** *sub-subst* = (·*l*) **and**
map = *image-mset* **and** *to-set* = *set-mset* **and** *sub-welltyped* = *literal.welltyped* **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*⟩

end

locale *nonground-inhabited-typing* =
nonground-typing $\mathcal{F}$ +
nonground-term-inhabited-typing $\mathcal{F}$
for $\mathcal{F}$ :: ('*f*, '*ty*) *fun-types*
begin

sublocale *atom*: *term-based-nonground-inhabited-typing-lifting* **where**
base-welltyped = *welltyped* **and** *map* = *map-uprod* **and** *to-set* = *set-uprod* **and**
sub-welltyped = *welltyped* **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* **and**
sub-subst = (·*t*) **and** *sub-vars* = *term.vars*
⟨*proof*⟩

sublocale *literal*: *term-based-nonground-inhabited-typing-lifting* **where**
base-welltyped = *welltyped* **and** *sub-vars* = *atom.vars* **and**
sub-subst = (·*a*) **and** *map* = *map-literal* **and** *to-set* = *set-literal* **and**
sub-welltyped = *atom.welltyped* **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*⟩

sublocale *clause*: *term-based-nonground-inhabited-typing-lifting* **where**
base-welltyped = *welltyped* **and**
sub-vars = *literal.vars* **and** *sub-subst* = (·*l*) **and** *map* = *image-mset* **and** *to-set* = *set-mset* **and**
sub-welltyped = *literal.welltyped* **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*⟩

end

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

lemmas *UniqI* $=$ *Uniq-I*

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

lemma *Uniq-implies-ex1*: $\exists_{\leq 1} x.\ P\ x \Longrightarrow P\ y \Longrightarrow \exists!x.\ P\ x$
⟨*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 *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 $x \in set\ X\ f\ x \notin set\ X\ map\ f\ X = X$
shows *False*
⟨*proof*⟩

lemma *Ball-singleton*: $(\forall\, x \in \{x\}.\ P\ x) \longleftrightarrow P\ x$
⟨*proof*⟩

end
theory *Grounded-Selection-Function*

imports
Nonground-Selection-Function
Nonground-Typing
HOL-Extra
begin

context *nonground-typing*
begin

abbreviation *select-subst-stability-on-clause* **where**
select-subst-stability-on-clause select select$_G$ C_G C $\mathcal{V}$ $\gamma \equiv$
$C \cdot \gamma =$ *clause.from-ground* $C_G \wedge$
select$_G$ $C_G =$ *clause.to-ground* $((\textit{select}\ C) \cdot \gamma) \wedge$
clause.is-welltyped-ground-instance C $\mathcal{V}$ γ

abbreviation *select-subst-stability-on* **where**
select-subst-stability-on select select$_G$ $N \equiv$
$\forall\, C_G \in \bigcup$ (*clause.welltyped-ground-instances* ‘ N). $\exists\, (C, \mathcal{V}) \in N.\ \exists \gamma.$
select-subst-stability-on-clause select select$_G$ C_G C $\mathcal{V}$ γ

lemma *obtain-subst-stable-on-select-grounding*:
fixes *select* :: $('f, 'v)$ *select*
obtains *select*$_G$ **where**
select-subst-stability-on select select$_G$ N
is-select-grounding select select$_G$
⟨*proof*⟩

end

locale *grounded-selection-function* =
nonground-selection-function select +
nonground-typing $\mathcal{F}$
for
select :: $('f, 'v ::$ *infinite*$)$ *atom clause* $\Rightarrow$ $('f, 'v)$ *atom clause* **and**
$\mathcal{F}$ :: $('f, 'ty)$ *fun-types* +
fixes *select*$_G$
assumes *select*$_G$: *is-select-grounding select select*$_G$
begin

abbreviation *subst-stability-on* **where**
subst-stability-on $N \equiv$ *select-subst-stability-on select select*$_G$ N

lemma *select*$_G$*-subset*: *select*$_G$ $C \subseteq\#$ C
⟨*proof*⟩

lemma *select*$_G$*-negative-literals*:
assumes $l_G \in\#$ *select*$_G$ C_G
shows *is-neg* l_G
⟨*proof*⟩

sublocale *ground*: *selection-function* $select_G$
⟨*proof*⟩

end

end
theory *Term-Rewrite-System*
imports *Ground-Context*
begin

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

lemma *compatible-with-gctxtD*:
compatible-with-gctxt I ⟹ (*t*, *t'*) ∈ *I* ⟹ (*ctxt*⟨*t*⟩$_G$, *ctxt*⟨*t'*⟩$_G$) ∈ *I*
⟨*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* $R \subseteq$ *rewrite-inside-gctxt* S
⟨*proof*⟩

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

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

lemma *converse-rewrite-steps*: (*rewrite-inside-gctxt* $R)^{-1}$ = *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: $(t1,\ t2) \in$ *rewrite-inside-gctxt* R **and**
ball-R-rhs-lt-lhs: $\bigwedge t1\ t2.\ (t1,\ t2) \in R \Longrightarrow t2 \prec_t t1$ **and**
compatible-with-gctxt: $\bigwedge t1\ t2\ ctxt.\ t2 \prec_t t1 \Longrightarrow ctxt\langle t2\rangle_G \prec_t ctxt\langle t1\rangle_G$
shows $t2 \prec_t t1$
⟨*proof*⟩

lemma *mem-rewrite-step-union-NF*:
assumes $(t,\ t') \in$ *rewrite-inside-gctxt* $(R1 \cup R2)$
$t \in NF$ (*rewrite-inside-gctxt* $R2$)
shows $(t,\ t') \in$ *rewrite-inside-gctxt* $R1$
⟨*proof*⟩

lemma *predicate-holds-of-mem-rewrite-inside-gctxt*:
assumes *rule-in*: $(t1,\ t2) \in$ *rewrite-inside-gctxt* R **and**
ball-P: $\bigwedge t1\ t2.\ (t1,\ t2) \in R \Longrightarrow P\ t1\ t2$ **and**
preservation: $\bigwedge t1\ t2\ ctxt\ \sigma.\ (t1,\ t2) \in R \Longrightarrow P\ t1\ t2 \Longrightarrow P\ ctxt\langle t1\rangle_G\ ctxt\langle t2\rangle_G$
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*]: $E \subseteq$ *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: $\bigwedge ctxt\ t\ t'.\ R\ t\ t' \Longrightarrow R\ ctxt\langle t\rangle_G\ ctxt\langle t'\rangle_G$ **and**
equations-subset-R: $\bigwedge x\ y.\ (x,\ y) \in E \Longrightarrow R\ y\ x$
shows *wf* $((rewrite\text{-}inside\text{-}gctxt\ E)^{-1})$
⟨*proof*⟩

end
theory *Entailment-Lifting*
imports *Abstract-Substitution.Functional-Substitution-Lifting*
begin

locale *entailment* =
based: *based-functional-substitution* **where** *base-subst* = *base-subst* **and** *vars* = *vars* +
base: *grounding* **where** *subst* = *base-subst* **and** *vars* = *base-vars* **and** *to-ground* = *base-to-ground* **and**
from-ground = *base-from-ground* **for**
vars :: $'expr \Rightarrow 'var\ set$ **and**
base-subst :: $'base \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'base$ **and**
base-to-ground :: $'base \Rightarrow 'base_G$ **and**
base-from-ground +
fixes *entails-def* :: $'expr \Rightarrow bool$ **and** I :: $('base_G \times 'base_G)\ set$
assumes
congruence: $\bigwedge expr\ \gamma\ var\ update.$
based.base.is-ground update $\Longrightarrow$
based.base.is-ground $(\gamma\ var) \Longrightarrow$
$(base\text{-}to\text{-}ground\ (\gamma\ var),\ base\text{-}to\text{-}ground\ update) \in I \Longrightarrow$
based.is-ground $(subst\ expr\ \gamma) \Longrightarrow$
entails-def $(subst\ expr\ (\gamma(var := update))) \Longrightarrow$
entails-def $(subst\ expr\ \gamma)$
begin

abbreviation *entails* ≡ *entails-def*

end

locale *symmetric-entailment* = *entailment* +
assumes *sym*: *sym* I
begin

lemma *symmetric-congruence*:
assumes
update-is-ground: *based.base.is-ground update* **and**
var-grounding: *based.base.is-ground* (γ *var*) **and**
var-update: (*base-to-ground* (γ *var*), *base-to-ground update*) $\in$ *I* **and**
expr-grounding: *based.is-ground* (*subst expr* γ)
shows
entails (*subst expr* (γ(*var* := *update*))) $\longleftrightarrow$ *entails* (*subst expr* γ)
⟨*proof*⟩

end

locale *symmetric-base-entailment* =
base-functional-substitution **where** *subst* = *subst* +
grounding **where** *subst* = *subst* **and** *to-ground* = *to-ground* **for**
subst :: *'base* $\Rightarrow$ (*'var* $\Rightarrow$ *'base*) $\Rightarrow$ *'base* (**infixl** · *70*) **and**
to-ground :: *'base* $\Rightarrow$ *'base*$_G$ +
fixes *I* :: (*'base*$_G$ $\times$ *'base*$_G$) *set*
assumes
sym: *sym I* **and**
congruence: $\bigwedge$*expr expr' update* γ *var*.
is-ground update $\Longrightarrow$
is-ground (γ *var*) $\Longrightarrow$
(*to-ground* (γ *var*), *to-ground update*) $\in$ *I* $\Longrightarrow$
is-ground (*expr* · γ) $\Longrightarrow$
(*to-ground* (*expr* · (γ(*var* := *update*))), *expr'*) $\in$ *I* $\Longrightarrow$
(*to-ground* (*expr* · γ), *expr'*) $\in$ *I*
begin

lemma *symmetric-congruence*:
assumes
update-is-ground: *is-ground update* **and**
var-grounding: *is-ground* (γ *var*) **and**
expr-grounding: *is-ground* (*expr* · γ) **and**
var-update: (*to-ground* (γ *var*), *to-ground update*) $\in$ *I*
shows (*to-ground* (*expr* · (γ(*var* := *update*))), *expr'*) $\in$ *I* $\longleftrightarrow$ (*to-ground* (*expr* · γ), *expr'*) $\in$ *I*
⟨*proof*⟩

lemma *simultaneous-congruence*:
assumes
update-is-ground: *is-ground update* **and**
var-grounding: *is-ground* (γ *var*) **and**
var-update: (*to-ground* (γ *var*), *to-ground update*) $\in$ *I* **and**
expr-grounding: *is-ground* (*expr* · γ) *is-ground* (*expr'* · γ)
shows
(*to-ground* (*expr* · (γ(*var* := *update*))), *to-ground* (*expr'* · (γ(*var* := *update*)))) $\in$ *I* $\longleftrightarrow$
(*to-ground* (*expr* · γ), *to-ground* (*expr'* · γ)) $\in$ *I*

⟨*proof*⟩

end

locale *entailment-lifting* =
  based-functional-substitution-lifting +
  finite-variables-lifting +
  sub: *symmetric-entailment*
  where *subst* = *sub-subst* **and** *vars* = *sub-vars* **and** *entails-def* = *sub-entails*
  for *sub-entails* +
  fixes
    is-negated :: *'d* ⇒ *bool* **and**
    empty :: *bool* **and**
    connective :: *bool* ⇒ *bool* ⇒ *bool* **and**
    entails-def
  assumes
    is-negated-subst: ⋀*expr σ*. *is-negated* (*subst expr σ*) ⟷ *is-negated expr* **and**
    entails-def: ⋀*expr*. *entails-def expr* ⟷
      (*if is-negated expr then Not else* (λ*x*. *x*))
        (*Finite-Set.fold connective empty* (*sub-entails* ‘ *to-set expr*))
begin

notation *sub-entails* ((⊨$_s$ -) [*50*] *50*)
notation *entails-def* ((⊨ -) [*50*] *50*)

sublocale *symmetric-entailment* **where** *subst* = *subst* **and** *vars* = *vars* **and** *entails-def* = *entails-def*
⟨*proof*⟩

end

locale *entailment-lifting-conj* = *entailment-lifting*
  where *connective* = (∧) **and** *empty* = *True*

locale *entailment-lifting-disj* = *entailment-lifting*
  where *connective* = (∨) **and** *empty* = *False*

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

lemma *comp-fun-idem-conj*: *comp-fun-idem-on X* (∧)
  ⟨*proof*⟩

lemma *comp-fun-idem-disj*: *comp-fun-idem-on X* (∨)
  ⟨*proof*⟩

lemma *fold-conj-insert* [*simp*]:

Finite-Set.fold (∧) *True* (*insert b B*) ⟷ *b* ∧ *Finite-Set.fold* (∧) *True B*
⟨*proof*⟩

lemma *fold-disj-insert* [*simp*]:
Finite-Set.fold (∨) *False* (*insert b B*) ⟷ *b* ∨ *Finite-Set.fold* (∨) *False B*
⟨*proof*⟩

end
theory *Nonground-Entailment*
imports
Nonground-Context
Nonground-Clause
Term-Rewrite-System
Entailment-Lifting
Fold-Extra
begin

4 Entailment

context *nonground-term*
begin

lemma *var-in-term*:
assumes *x* ∈ *vars t*
obtains *c* **where** *t* = *c*⟨*Var x*⟩
⟨*proof*⟩

lemma *vars-term-ms-count*:
assumes *is-ground t*
shows
size {#*x'* ∈# *vars-term-ms c*⟨*Var x*⟩. *x'* = *x*#} = *Suc* (*size* {#*x'* ∈# *vars-term-ms c*⟨*t*⟩. *x'* = *x*#})
⟨*proof*⟩

end

context *nonground-clause*
begin

lemma *not-literal-entails* [*simp*]:
¬ *upair* ‘ *I* ⊨*l Neg a* ⟷ *upair* ‘ *I* ⊨*l Pos a*
¬ *upair* ‘ *I* ⊨*l Pos a* ⟷ *upair* ‘ *I* ⊨*l Neg a*
⟨*proof*⟩

lemmas *literal-entails-unfolds* =
not-literal-entails true-lit-simps

end

locale *clause-entailment* = *nonground-clause* +
  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 *symmetric-context-congruence*:
  assumes $(t,\ t') \in I$
  shows $(c\langle t\rangle_G,\ t'') \in I \longleftrightarrow (c\langle t'\rangle_G,\ t'') \in I$
  ⟨*proof*⟩

lemma *symmetric-upair-context-congruence*:
  assumes *Upair t t'* ∈ *upair* ‘ *I*
  shows *Upair* $c\langle t\rangle_G$ *t''* ∈ *upair* ‘ *I* ⟷ *Upair* $c\langle t'\rangle_G$ *t''* ∈ *upair* ‘ *I*
  ⟨*proof*⟩

lemma *upair-compatible-with-gctxtI* [*intro*]:
  Upair t t' ∈ *upair* ‘ *I* ⟹ *Upair* $c\langle t\rangle_G$ $c\langle t'\rangle_G$ ∈ *upair* ‘ *I*
  ⟨*proof*⟩

sublocale *term*: *symmetric-base-entailment* **where** *vars* = *term.vars* :: (*'f*, *'v*) *term* ⇒ *'v set* **and**
  id-subst = *Var* **and** *comp-subst* = (⊙) **and** *subst* = (·*t*) **and** *to-ground* = *term.to-ground* **and**
  from-ground = *term.from-ground*
⟨*proof*⟩

sublocale *atom*: *symmetric-entailment*
  where *comp-subst* = (⊙) **and** *id-subst* = *Var*
    and *base-subst* = (·*t*) **and** *base-vars* = *term.vars* **and** *subst* = (·*a*) **and** *vars* = *atom.vars*
  and *base-to-ground* = *term.to-ground* **and** *base-from-ground* = *term.from-ground* **and** *I* = *I*
    and *entails-def* = λ*a*. *atom.to-ground a* ∈ *upair* ‘ *I*
⟨*proof*⟩

sublocale *literal*: *entailment-lifting-conj*
  where *comp-subst* = (⊙) **and** *id-subst* = *Var*
    and *base-subst* = (·*t*) **and** *base-vars* = *term.vars* **and** *sub-subst* = (·*a*) **and** *sub-vars* = *atom.vars*
  and *base-to-ground* = *term.to-ground* **and** *base-from-ground* = *term.from-ground* **and** *I* = *I*
    and *sub-entails* = *atom.entails* **and** *map* = *map-literal* **and** *to-set* = *set-literal*
    and *is-negated* = *is-neg* **and** *entails-def* = λ*l*. *upair* ‘ *I* ⊨*l* *literal.to-ground l*
⟨*proof*⟩

sublocale *clause*: *entailment-lifting-disj*

where *comp-subst* = $(\odot)$ **and** *id-subst* = *Var*
and *base-subst* = $(\cdot t)$ **and** *base-vars* = *term.vars*
and *base-to-ground* = *term.to-ground* **and** *base-from-ground* = *term.from-ground*
and $I = I$
and *sub-subst* = $(\cdot l)$ **and** *sub-vars* = *literal.vars* **and** *sub-entails* = *literal.entails*
and *map* = *image-mset* **and** *to-set* = *set-mset* **and** *is-negated* = λ-. *False*
and *entails-def* = $\lambda C.$ *upair* ‘ *I* $\models$ *clause.to-ground C*
⟨*proof*⟩

lemma *literal-compatible-with-gctxtI* [*intro*]:
literal.entails $(t \approx t') \Longrightarrow$ *literal.entails* $(c\langle t\rangle \approx c\langle t'\rangle)$
⟨*proof*⟩

lemma *symmetric-literal-context-congruence*:
assumes *Upair t t'* $\in$ *upair* ‘ *I*
shows
upair ‘ *I* $\models l$ $c\langle t\rangle_G \approx t'' \longleftrightarrow$ *upair* ‘ *I* $\models l$ $c\langle t'\rangle_G \approx t''$
upair ‘ *I* $\models l$ $c\langle t\rangle_G$!$\approx t'' \longleftrightarrow$ *upair* ‘ *I* $\models l$ $c\langle t'\rangle_G$!$\approx t''$
⟨*proof*⟩

end

end
theory *Nonground-Inference*
imports *Nonground-Clause Nonground-Typing*
begin

locale *nonground-inference* = *nonground-clause*
begin

sublocale *inference*: *term-based-lifting* **where**
sub-subst = *clause.subst* **and** *sub-vars* = *clause.vars* **and** *map* = *map-inference*
and
to-set = *set-inference* **and** *sub-to-ground* = *clause.to-ground* **and**
sub-from-ground = *clause.from-ground* **and** *to-ground-map* = *map-inference* **and**
from-ground-map = *map-inference* **and** *ground-map* = *map-inference* **and** *to-set-ground*
= *set-inference*
⟨*proof*⟩

notation *inference.subst* (**infixl** $\cdot\iota$ *67*)

lemma *vars-inference* [*simp*]:
inference.vars (*Infer Ps C*) = $\bigcup$ (*clause.vars* ‘ *set Ps*) $\cup$ *clause.vars C*
⟨*proof*⟩

lemma *subst-inference* [*simp*]:
Infer Ps C $\cdot\iota$ σ = *Infer* (*map* ($\lambda P.$ $P \cdot \sigma$) *Ps*) ($C \cdot \sigma$)
⟨*proof*⟩

lemma *inference-from-ground-clause-from-ground* [*simp*]:
inference.from-ground (*Infer Ps C*) = *Infer* (*map clause.from-ground Ps*) (*clause.from-ground C*)
⟨*proof*⟩

lemma *inference-to-ground-clause-to-ground* [*simp*]:
inference.to-ground (*Infer Ps C*) = *Infer* (*map clause.to-ground Ps*) (*clause.to-ground C*)
⟨*proof*⟩

lemma *inference-is-ground-clause-is-ground* [*simp*]:
inference.is-ground (*Infer Ps C*) ⟷ *list-all clause.is-ground Ps* ∧ *clause.is-ground C*
⟨*proof*⟩

end

end
theory *Restricted-Order*
imports *Main*
begin

5 Restricted Orders

locale *relation-restriction* =
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool* **and** *lift* :: *'b* ⇒ *'a*
assumes *inj-lift* [*intro*]: *inj lift*
begin

definition R_r :: *'b* ⇒ *'b* ⇒ *bool* **where**
R_r *b b'* ≡ *R* (*lift b*) (*lift b'*)

end

5.1 Strict Orders

locale *strict-order* =
fixes
less :: *'a* ⇒ *'a* ⇒ *bool* (**infix** ≺ *50*)
assumes
transp [*intro*]: *transp* (≺) **and**
asymp [*intro*]: *asymp* (≺)
begin

abbreviation *less-eq* **where** *less-eq* ≡ (≺)$^{==}$

notation *less-eq* (**infix** ⪯ *50*)

sublocale *order* (⪯) (≺)

⟨*proof*⟩

end

locale *strict-order-restriction* =
  strict-order +
  relation-restriction **where** $R = (\prec)$
begin

abbreviation $less_r \equiv R_r$

lemmas $less_r$*-def* = R_r*-def*

notation $less_r$ (**infix** $\prec_r$ *50*)

sublocale *restriction*: *strict-order* $(\prec_r)$
  ⟨*proof*⟩

abbreviation $less\text{-}eq_r \equiv$ *restriction.less-eq*
notation $less\text{-}eq_r$ (**infix** $\preceq_r$ *50*)

end

5.2 Wellfounded Strict Orders

locale *restricted-wellfounded-strict-order* = *strict-order* +
  fixes *restriction*
  assumes *wfp* [*intro*]: *wfp-on restriction* $(\prec)$

locale *wellfounded-strict-order* =
  restricted-wellfounded-strict-order **where** *restriction* = *UNIV*

locale *wellfounded-strict-order-restriction* =
  strict-order-restriction +
  restricted-wellfounded-strict-order **where** *restriction* = *range lift* **and** *less* = $(\prec)$
begin

sublocale *wellfounded-strict-order* $(\prec_r)$
⟨*proof*⟩

end

5.3 Total Strict Orders

locale *restricted-total-strict-order* = *strict-order* +
  fixes *restriction*
  assumes *totalp* [*intro*]: *totalp-on restriction* $(\prec)$
begin

lemma *restricted-not-le*:

assumes $a \in \mathit{restriction}\ b \in \mathit{restriction} \neg\ b \prec a$
shows $a \preceq b$
⟨*proof*⟩

end

locale *total-strict-order* =
restricted-total-strict-order **where** *restriction* = *UNIV*
begin

sublocale *linorder* $(\preceq)$ $(\prec)$
⟨*proof*⟩

end

locale *total-strict-order-restriction* =
strict-order-restriction +
restricted-total-strict-order **where** *restriction* = *range lift* **and** *less* = $(\prec)$
begin

sublocale *total-strict-order* $(\prec_r)$
⟨*proof*⟩

end

locale *restricted-wellfounded-total-strict-order* =
restricted-wellfounded-strict-order + *restricted-total-strict-order*

end
theory *Context-Compatible-Order*
imports
Ground-Context
Restricted-Order
begin

locale *restriction-restricted* =
fixes *restriction context-restriction restricted restricted-context*
assumes
restricted:
$\bigwedge t.\ t \in \mathit{restriction} \longleftrightarrow \mathit{restricted}\ t$
$\bigwedge c.\ c \in \mathit{context\text{-}restriction} \longleftrightarrow \mathit{restricted\text{-}context}\ c$

locale *restricted-context-compatibility* =
restriction-restricted +
fixes *R Fun*
assumes
context-compatible [*simp*]:
$\bigwedge c\ t_1\ t_2.$
restricted $t_1 \Longrightarrow$

restricted $t_2 \Longrightarrow$
restricted-context $c \Longrightarrow$
R (*Fun*$\langle c;t_1\rangle$) (*Fun*$\langle c;t_2\rangle$) $\longleftrightarrow$ R t_1 t_2

locale *context-compatibility* = *restricted-context-compatibility* **where**
restriction = *UNIV* **and** *context-restriction* = *UNIV* **and** *restricted* = λ-. *True*
and
restricted-context = λ-. *True*
begin

lemma *context-compatibility* [*simp*]: R (*Fun*$\langle c;t_1\rangle$) (*Fun*$\langle c;t_2\rangle$) $\longleftrightarrow$ R t_1 t_2
⟨*proof*⟩

end

locale *context-compatible-restricted-order* =
restricted-total-strict-order +
restriction-restricted +
fixes *Fun*
assumes *less-context-compatible*:
$\bigwedge c$ t_1 t_2.
restricted $t_1 \Longrightarrow$
restricted $t_2 \Longrightarrow$
restricted-context $c \Longrightarrow$
$t_1 \prec t_2 \Longrightarrow$
Fun$\langle c;t_1\rangle \prec$ *Fun*$\langle c;t_2\rangle$
begin

sublocale *restricted-context-compatibility* **where** $R = (\prec)$
⟨*proof*⟩

sublocale *less-eq*: *restricted-context-compatibility* **where** $R = (\preceq)$
⟨*proof*⟩

lemma *context-less-term-lesseq*:
assumes
restricted t
restricted t'
restricted-context c
restricted-context c'
$\bigwedge t$. *restricted* $t \Longrightarrow$ *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t\rangle$
$t \preceq t'$
shows *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t'\rangle$
⟨*proof*⟩

lemma *context-lesseq-term-less*:
assumes
restricted t
restricted t'

restricted-context c
restricted-context c′
$\bigwedge t$. *restricted* $t \Longrightarrow$ *Fun*$\langle c;t\rangle \preceq$ *Fun*$\langle c';t\rangle$
$t \prec t'$
shows *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t'\rangle$
⟨*proof*⟩

end

locale *context-compatible-order* =
total-strict-order +
fixes *Fun*
assumes *less-context-compatible*: $t_1 \prec t_2 \Longrightarrow$ *Fun*$\langle c;t_1\rangle \prec$ *Fun*$\langle c;t_2\rangle$
begin

sublocale *restricted*: *context-compatible-restricted-order* **where**
restriction = *UNIV* **and** *context-restriction* = *UNIV* **and** *restricted* = λ-. *True*
and
restricted-context = λ-. *True*
⟨*proof*⟩

sublocale *context-compatibility* ($\prec$)
⟨*proof*⟩

sublocale *less-eq*: *context-compatibility* ($\preceq$)
⟨*proof*⟩

lemma *context-less-term-lesseq*:
assumes
$\bigwedge t$. *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t\rangle$
$t \preceq t'$
shows *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t'\rangle$
⟨*proof*⟩

lemma *context-lesseq-term-less*:
assumes
$\bigwedge t$. *Fun*$\langle c;t\rangle \preceq$ *Fun*$\langle c';t\rangle$
$t \prec t'$
shows *Fun*$\langle c;t\rangle \prec$ *Fun*$\langle c';t'\rangle$
⟨*proof*⟩

end

end
theory *Term-Order-Notation*
imports *Main*
begin

locale *term-order-notation* =

fixes $less_t :: 't \Rightarrow 't \Rightarrow bool$
begin

notation $less_t$ (**infix** $\prec_t$ *50*)

abbreviation *less-eq*$_t \equiv (\prec_t)^{==}$

notation *less-eq*$_t$ (**infix** $\preceq_t$ *50*)

end

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

lemma *reflclp-iff*: $\bigwedge R\ x\ y.\ R^{==}\ x\ y \longleftrightarrow R\ x\ y \vee x = y$
⟨*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 $\bigwedge z.\ (x,\ z) \in A^* \Longrightarrow z \notin Domain\ B$
shows $(x,\ y) \in (A \cup B)^* \longleftrightarrow (x,\ y) \in A^*$
⟨*proof*⟩

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

end
theory *Ground-Term-Order*
imports
Ground-Context
Context-Compatible-Order
Term-Order-Notation

Transitive-Closure-Extra
begin

locale *context-compatible-ground-order* = *context-compatible-order* **where** *Fun* = *GFun*

locale *subterm-property* =
  strict-order **where** *less* = $less_t$
  for $less_t$ :: *'f gterm* ⇒ *'f gterm* ⇒ *bool* +
  assumes
    subterm-property [*simp*]: $\bigwedge t\ c.\ c \neq \Box \Longrightarrow less_t\ t\ c\langle t\rangle_G$
begin

interpretation *term-order-notation*⟨*proof*⟩

lemma *less-eq-subterm-property*: $t \preceq_t c\langle t\rangle_G$
  ⟨*proof*⟩

end

locale *ground-term-order* =
  wellfounded-strict-order $less_t$ +
  total-strict-order $less_t$ +
  context-compatible-ground-order $less_t$ +
  subterm-property $less_t$
  for $less_t$ :: *'f gterm* ⇒ *'f gterm* ⇒ *bool*
begin

interpretation *term-order-notation*⟨*proof*⟩

end

end
theory *Grounded-Order*
  imports
    Restricted-Order
    Abstract-Substitution.Functional-Substitution-Lifting
begin

6 Orders with ground restrictions

locale *grounded-order* =
  strict-order **where** *less* = *less* +
  grounding **where** *vars* = *vars*
for
  less :: *'expr* ⇒ *'expr* ⇒ *bool* (**infix** ‹≺› *50*) **and**
  vars :: *'expr* ⇒ *'var set*
begin

sublocale *strict-order-restriction* **where** *lift* = *from-ground*
⟨*proof*⟩

abbreviation $less_G \equiv less_r$
lemmas $less_G$-*def* = $less_r$-*def*
notation $less_G$ (**infix** $\prec_G$ *50*)

abbreviation $less\text{-}eq_G \equiv less\text{-}eq_r$
notation $less\text{-}eq_G$ (**infix** $\preceq_G$ *50*)

lemma $to\text{-}ground\text{-}less_r$ [*simp*]:
assumes *is-ground e* **and** *is-ground e'*
shows *to-ground e* $\prec_G$ *to-ground e'* $\longleftrightarrow$ $e \prec e'$
⟨*proof*⟩

lemma $to\text{-}ground\text{-}less\text{-}eq_r$ [*simp*]:
assumes *is-ground e* **and** *is-ground e'*
shows *to-ground e* $\preceq_G$ *to-ground e'* $\longleftrightarrow$ $e \preceq e'$
⟨*proof*⟩

lemma $less\text{-}eq_r$-*from-ground* [*simp*]:
$e_G \preceq_G e_G' \longleftrightarrow$ *from-ground* $e_G \preceq$ *from-ground* e_G'
⟨*proof*⟩

end

locale *grounded-restricted-total-strict-order* =
order: *restricted-total-strict-order* **where** *restriction* = *range from-ground* + *grounded-order*
begin

sublocale *total-strict-order-restriction* **where** *lift* = *from-ground*
⟨*proof*⟩

lemma *not-less-eq* [*simp*]:
assumes *is-ground expr* **and** *is-ground expr'*
shows ¬ *order.less-eq expr' expr* $\longleftrightarrow$ *expr* $\prec$ *expr'*
⟨*proof*⟩

end

locale *grounded-restricted-wellfounded-strict-order* =
restricted-wellfounded-strict-order **where** *restriction* = *range from-ground* + *grounded-order*
begin

sublocale *wellfounded-strict-order-restriction* **where** *lift* = *from-ground*
⟨*proof*⟩

end

6.1 Ground substitution stability

locale *ground-subst-stability* = *grounding* +
  fixes R
  assumes
    ground-subst-stability:
      $\bigwedge expr_1\ expr_2\ \gamma.$
        is-ground $(expr_1 \cdot \gamma) \Longrightarrow$
        is-ground $(expr_2 \cdot \gamma) \Longrightarrow$
        $R\ expr_1\ expr_2 \Longrightarrow$
        $R\ (expr_1 \cdot \gamma)\ (expr_2 \cdot \gamma)$

locale *ground-subst-stable-grounded-order* =
  grounded-order +
  ground-subst-stability **where** $R = (\prec)$
begin

sublocale *less-eq*: *ground-subst-stability* **where** $R = (\preceq)$
  ⟨*proof*⟩

lemma *ground-less-not-less-eq*:
  assumes
    grounding: *is-ground* $(expr_1 \cdot \gamma)$ *is-ground* $(expr_2 \cdot \gamma)$ **and**
    less: $expr_1 \cdot \gamma \prec expr_2 \cdot \gamma$
  shows
    $\neg\ expr_2 \preceq expr_1$
  ⟨*proof*⟩

end

6.2 Substitution update stability

locale *subst-update-stability* =
  based-functional-substitution +
  fixes *base-R* R
  assumes
    subst-update-stability:
      $\bigwedge update\ x\ \gamma\ expr.$
        base.is-ground update $\Longrightarrow$
        base-R update $(\gamma\ x) \Longrightarrow$
        is-ground $(expr \cdot \gamma) \Longrightarrow$
        $x \in vars\ expr \Longrightarrow$
        $R\ (expr \cdot \gamma(x := update))\ (expr \cdot \gamma)$

locale *base-subst-update-stability* =
  base-functional-substitution +

subst-update-stability **where** *base-R* = *R* **and** *base-subst* = *subst* **and** *base-vars* = *vars*

locale *subst-update-stable-grounded-order* =
grounded-order + *subst-update-stability* **where** *R* = *less* **and** *base-R* = *base-less*
for *base-less*
begin

sublocale *less-eq*: *subst-update-stability*
where *base-R* = *base-less*$^{==}$ **and** *R* = *less*$^{==}$
⟨*proof*⟩

end

locale *base-subst-update-stable-grounded-order* =
base-subst-update-stability **where** *R* = *less* +
subst-update-stable-grounded-order **where**
base-less = *less* **and** *base-subst* = *subst* **and** *base-vars* = *vars*

end
theory *Multiset-Extension*
imports
Restricted-Order
Multiset-Extra
begin

7 Multiset Extensions

locale *multiset-extension* = *order*: *strict-order* +
fixes *to-mset* :: $'b \Rightarrow 'a$ *multiset*
begin

definition *multiset-extension* :: $'b \Rightarrow 'b \Rightarrow$ *bool* **where**
multiset-extension b1 b2 ≡ *multp* (≺) (*to-mset b1*) (*to-mset b2*)

notation *multiset-extension* (**infix** $\prec_m$ *50*)

sublocale *strict-order* ($\prec_m$)
⟨*proof*⟩

notation *less-eq* (**infix** $\preceq_m$ *50*)

end

7.1 Wellfounded Multiset Extensions

locale *wellfounded-multiset-extension* =
order: *wellfounded-strict-order* +

multiset-extension
begin

sublocale *wellfounded-strict-order* $(\prec_m)$
⟨*proof*⟩

end

7.2 Total Multiset Extensions

locale *restricted-total-multiset-extension* =
base: *restricted-total-strict-order* +
multiset-extension +
assumes *inj-on-to-mset*: *inj-on to-mset* {*b*. *set-mset* (*to-mset b*) ⊆ *restriction*}
begin

sublocale *restricted-total-strict-order* $(\prec_m)$ {*b*. *set-mset* (*to-mset b*) ⊆ *restriction*}
⟨*proof*⟩

end

locale *total-multiset-extension* =
order: *total-strict-order* +
multiset-extension +
assumes *inj-to-mset*: *inj to-mset*
begin

sublocale *restricted-total-multiset-extension* **where** *restriction* = *UNIV*
⟨*proof*⟩

sublocale *total-strict-order* $(\prec_m)$
⟨*proof*⟩

end

locale *total-wellfounded-multiset-extension* =
wellfounded-multiset-extension + *total-multiset-extension*

end
theory *Grounded-Multiset-Extension*
imports *Grounded-Order Multiset-Extension*
begin

8 Grounded Multiset Extensions

locale *functional-substitution-multiset-extension* =
sub: *strict-order* **where** *less* = $(\prec)$ :: *'sub* ⇒ *'sub* ⇒ *bool* +
multiset-extension **where** *to-mset* = *to-mset* +
functional-substitution-lifting **where** *id-subst* = *id-subst* **and** *to-set* = *to-set*

for
to-mset :: *'expr* ⇒ *'sub multiset* **and**
id-subst :: *'var* ⇒ *'base* **and**
to-set :: *'expr* ⇒ *'sub set* +
assumes

to-mset-to-set: ⋀*expr*. *set-mset* (*to-mset expr*) = *to-set expr* **and**
to-mset-map: ⋀*f b*. *to-mset* (*map f b*) = *image-mset f* (*to-mset b*) **and**
inj-to-mset: *inj to-mset*
begin

no-notation *less-eq* (**infix** ≼ *50*)
notation *sub.less-eq* (**infix** ≼ *50*)

lemma *lesseq-if-all-lesseq*:
assumes ∀ *sub* ∈# *to-mset expr*. *sub* ⋅$_s$ *σ'* ≼ *sub* ⋅$_s$ *σ*
shows *expr* ⋅ *σ'* ≼$_m$ *expr* ⋅ *σ*
⟨*proof*⟩

lemma *less-if-all-lesseq-ex-less*:
assumes
∀ *sub*∈#*to-mset expr*. *sub* ⋅$_s$ *σ'* ≼ *sub* ⋅$_s$ *σ*
∃ *sub*∈#*to-mset expr*. *sub* ⋅$_s$ *σ'* ≺ *sub* ⋅$_s$ *σ*
shows
expr ⋅ *σ'* ≺$_m$ *expr* ⋅ *σ*
⟨*proof*⟩

end

locale *grounded-multiset-extension* =
grounding-lifting **where**
id-subst = *id-subst* :: *'var* ⇒ *'base* **and** *to-set* = *to-set* :: *'expr* ⇒ *'sub set* **and**
to-set-ground = *to-set-ground* +
functional-substitution-multiset-extension **where** *to-mset* = *to-mset*
for
to-mset :: *'expr* ⇒ *'sub multiset* **and**
to-set-ground :: *'expr*$_G$ ⇒ *'sub*$_G$ *set*
begin

sublocale *strict-order-restriction* (≺$_m$) *from-ground*
⟨*proof*⟩

end

locale *total-grounded-multiset-extension* =
grounded-multiset-extension +
sub: *total-strict-order-restriction* **where** *lift* = *sub-from-ground*

begin

sublocale *total-strict-order-restriction* $(\prec_m)$ *from-ground*
⟨*proof*⟩

end

locale *based-grounded-multiset-extension* =
based-functional-substitution-lifting **where** *base-vars* = *base-vars* +
grounded-multiset-extension +
base: *strict-order* **where** *less* = *base-less*
for
base-vars :: *'base* ⇒ *'var set* **and**
base-less :: *'base* ⇒ *'base* ⇒ *bool*

8.1 Ground substitution stability

locale *ground-subst-stable-total-multiset-extension* =
grounded-multiset-extension +
sub: *ground-subst-stable-grounded-order* **where**
less = *less* **and** *subst* = *sub-subst* **and** *vars* = *sub-vars* **and** *from-ground* = *sub-from-ground* **and**
to-ground = *sub-to-ground*
begin

sublocale *ground-subst-stable-grounded-order* **where**
less = $(\prec_m)$ **and** *subst* = *subst* **and** *vars* = *vars* **and** *from-ground* = *from-ground* **and**
to-ground = *to-ground*
⟨*proof*⟩

end

8.2 Substitution update stability

locale *subst-update-stable-multiset-extension* =
based-grounded-multiset-extension +
sub: *subst-update-stable-grounded-order* **where**
vars = *sub-vars* **and** *subst* = *sub-subst* **and** *to-ground* = *sub-to-ground* **and**
from-ground = *sub-from-ground*
begin

no-notation *less-eq* (**infix** $\preceq$ *50*)

sublocale *subst-update-stable-grounded-order* **where**
less = $(\prec_m)$ **and** *vars* = *vars* **and** *subst* = *subst* **and** *from-ground* = *from-ground* **and**
to-ground = *to-ground*
⟨*proof*⟩

end

end
theory *Maximal-Literal*
imports
Clausal-Calculus-Extra
Min-Max-Least-Greatest.Min-Max-Least-Greatest-Multiset
Restricted-Order
begin

locale *maximal-literal* = *order*: *strict-order* **where** *less* = *less*
for *less* :: *'a literal* ⇒ *'a literal* ⇒ *bool*
begin

abbreviation *is-maximal* :: *'a literal* ⇒ *'a clause* ⇒ *bool* **where**
is-maximal l C ≡ *order.is-maximal-in-mset C l*

abbreviation *is-strictly-maximal* :: *'a literal* ⇒ *'a clause* ⇒ *bool* **where**
is-strictly-maximal l C ≡ *order.is-strictly-maximal-in-mset C l*

lemmas *is-maximal-def* = *order.is-maximal-in-mset-iff*

lemmas *is-strictly-maximal-def* = *order.is-strictly-maximal-in-mset-iff*

lemmas *is-maximal-if-is-strictly-maximal* = *order.is-maximal-in-mset-if-is-strictly-maximal-in-mset*

lemma *maximal-in-clause*:
assumes *is-maximal l C*
shows *l* ∈# *C*
⟨*proof*⟩

lemma *strictly-maximal-in-clause*:
assumes *is-strictly-maximal l C*
shows *l* ∈# *C*
⟨*proof*⟩

lemma *is-maximal-not-empty* [*intro*]: *is-maximal l C* ⟹ *C* ≠ {#}
⟨*proof*⟩

lemma *is-strictly-maximal-not-empty* [*intro*]: *is-strictly-maximal l C* ⟹ *C* ≠ {#}
⟨*proof*⟩

end

end
theory *Term-Order-Lifting*
imports

Grounded-Multiset-Extension
Maximal-Literal
Term-Order-Notation
begin

locale *restricted-term-order-lifting* =
term.order: *restricted-wellfounded-total-strict-order* **where** *less* = $less_t$
for $less_t$ :: $'t \Rightarrow 't \Rightarrow bool$ +
fixes *literal-to-mset* :: $'a$ *literal* $\Rightarrow$ $'t$ *multiset*
assumes *inj-literal-to-mset*: *inj literal-to-mset*
begin

sublocale *term-order-notation*⟨*proof*⟩

abbreviation *literal-order-restriction* **where**
literal-order-restriction ≡ {*b*. *set-mset* (*literal-to-mset b*) ⊆ *restriction*}

sublocale *literal.order*: *restricted-total-multiset-extension* **where**
less = ($\prec_t$) **and** *to-mset* = *literal-to-mset*
⟨*proof*⟩

notation *literal.order.multiset-extension* (**infix** $\prec_l$ *50*)
notation *literal.order.less-eq* (**infix** $\preceq_l$ *50*)

lemmas $less_l$*-def* = *literal.order.multiset-extension-def*

sublocale *maximal-literal* ($\prec_l$)
⟨*proof*⟩

sublocale *clause.order*: *restricted-total-multiset-extension* **where**
less = ($\prec_l$) **and** *to-mset* = $\lambda x.\ x$ **and** *restriction* = *literal-order-restriction*
⟨*proof*⟩

notation *clause.order.multiset-extension* (**infix** $\prec_c$ *50*)
notation *clause.order.less-eq* (**infix** $\preceq_c$ *50*)

lemmas $less_c$*-def* = *clause.order.multiset-extension-def*

end

locale *term-order-lifting* =
restricted-term-order-lifting **where** *restriction* = *UNIV* +
term.order: *wellfounded-strict-order* $less_t$ +
term.order: *total-strict-order* $less_t$
begin

sublocale *literal.order*: *total-wellfounded-multiset-extension* **where**
less = ($\prec_t$) **and** *to-mset* = *literal-to-mset*
⟨*proof*⟩

sublocale *clause.order*: *total-wellfounded-multiset-extension* **where**
less = $(\prec_l)$ **and** *to-mset* = $\lambda x.\ x$
⟨*proof*⟩

end

end
theory *Ground-Order*
imports *Ground-Term-Order Term-Order-Lifting*
begin

locale *ground-order* =
term.order: *ground-term-order* +
term-order-lifting

locale *ground-order-with-equality* =
term.order: *ground-term-order*
begin

sublocale *ground-order*
where *literal-to-mset* = *mset-lit*
⟨*proof*⟩

end

end
theory *Nonground-Term-Order*
imports
Nonground-Term
Nonground-Context
Ground-Order
begin

locale *ground-context-compatible-order* =
nonground-term-with-context +
restricted-total-strict-order **where** *restriction* = *range term.from-ground* +
assumes *ground-context-compatibility*:
$\bigwedge c\ t_1\ t_2.$
term.is-ground $t_1 \Longrightarrow$
term.is-ground $t_2 \Longrightarrow$
context.is-ground $c \Longrightarrow$
$t_1 \prec t_2 \Longrightarrow$
$c\langle t_1 \rangle \prec c\langle t_2 \rangle$
begin

sublocale *context-compatible-restricted-order* **where**
restriction = *range term.from-ground* **and** *context-restriction* = *range context.from-ground*

and
Fun = *Fun* **and** *restricted* = *term.is-ground* **and** *restricted-context* = *context.is-ground*
⟨*proof*⟩

end

locale *ground-subterm-property* =
nonground-term-with-context +
fixes *R*
assumes *ground-subterm-property*:
⋀t_G c_G.
term.is-ground t_G ⟹
context.is-ground c_G ⟹
$c_G \neq \Box$ ⟹
R t_G $c_G\langle t_G\rangle$

locale *base-grounded-order* =
order: *base-subst-update-stable-grounded-order* +
order: *grounded-restricted-total-strict-order* +
order: *grounded-restricted-wellfounded-strict-order* +
order: *ground-subst-stable-grounded-order* +
grounding

locale *nonground-term-order* =
nonground-term-with-context +
order: *restricted-wellfounded-total-strict-order* **where**
less = $less_t$ **and** *restriction* = *range term.from-ground* +
order: *ground-subst-stability* **where** *R* = $less_t$ **and** *comp-subst* = (⊙) **and** *subst* = $(\cdot t)$ **and**
vars = *term.vars* **and** *id-subst* = *Var* **and** *to-ground* = *term.to-ground* **and**
from-ground = *term.from-ground* +
order: *ground-context-compatible-order* **where** *less* = $less_t$ +
order: *ground-subterm-property* **where** *R* = $less_t$
for $less_t$ :: ('*f*, '*v*) *Term.term* ⇒ ('*f*, '*v*) *Term.term* ⇒ *bool*
begin

interpretation *term-order-notation*⟨*proof*⟩

sublocale *base-grounded-order* **where**
comp-subst = (⊙) **and** *subst* = $(\cdot t)$ **and** *vars* = *term.vars* **and** *id-subst* = *Var* **and**
to-ground = *term.to-ground* **and** *from-ground* = *term.from-ground* **and** *less* = $(\prec_t)$
⟨*proof*⟩

notation *order.less*$_G$ (**infix** $\prec_{tG}$ *50*)
notation *order.less-eq*$_G$ (**infix** $\preceq_{tG}$ *50*)

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

end

end
theory *Nonground-Order*
imports
Nonground-Clause
Nonground-Term-Order
Term-Order-Lifting
begin

9 Nonground Order

locale *nonground-order-lifting* =
grounding-lifting +
order: *total-grounded-multiset-extension* +
order: *ground-subst-stable-total-multiset-extension* +
order: *subst-update-stable-multiset-extension*
begin

sublocale *order*: *grounded-restricted-total-strict-order* **where**
less = *order.multiset-extension* **and** *subst* = *subst* **and** *vars* = *vars* **and** *to-ground* = *to-ground* **and**
from-ground = *from-ground*
⟨*proof*⟩

end

locale *nonground-term-based-order-lifting* =
term: *nonground-term* +
nonground-order-lifting **where**
id-subst = *Var* **and** *comp-subst* = ($\odot$) **and** *base-vars* = *term.vars* **and** *base-less* = $less_t$ **and**
base-subst = ($\cdot t$)
for $less_t$

locale *nonground-equality-order* =
nonground-clause +
term: *nonground-term-order*
begin

sublocale *restricted-term-order-lifting* **where**
restriction = *range term.from-ground* **and** *literal-to-mset* = *mset-lit*
⟨*proof*⟩

notation *term.order.less*$_G$ (**infix** $\prec_{tG}$ *50*)
notation *term.order.less-eq*$_G$ (**infix** $\preceq_{tG}$ *50*)

sublocale *literal*: *nonground-term-based-order-lifting* **where**
less = *less*$_t$ **and** *sub-subst* = $(\cdot t)$ **and** *sub-vars* = *term.vars* **and** *sub-to-ground* = *term.to-ground* **and**
sub-from-ground = *term.from-ground* **and** *map* = *map-uprod-literal* **and** *to-set* = *uprod-literal-to-set* **and**
to-ground-map = *map-uprod-literal* **and** *from-ground-map* = *map-uprod-literal* **and**
ground-map = *map-uprod-literal* **and** *to-set-ground* = *uprod-literal-to-set* **and** *to-mset* = *mset-lit*
rewrites
$\bigwedge l\ \sigma$. *functional-substitution-lifting.subst* $(\cdot t)$ *map-uprod-literal* $l\ \sigma$ = *literal.subst* $l\ \sigma$ **and**
$\bigwedge l$. *functional-substitution-lifting.vars term.vars uprod-literal-to-set* l = *literal.vars* l **and**
$\bigwedge l_G$. *grounding-lifting.from-ground term.from-ground map-uprod-literal* l_G
= *literal.from-ground* l_G **and**
$\bigwedge l$. *grounding-lifting.to-ground term.to-ground map-uprod-literal* l = *literal.to-ground* l
⟨*proof*⟩

notation *literal.order.less*$_G$ (**infix** $\prec_{lG}$ *50*)
notation *literal.order.less-eq*$_G$ (**infix** $\preceq_{lG}$ *50*)

sublocale *clause*: *nonground-term-based-order-lifting* **where**
less = $(\prec_l)$ **and** *sub-subst* = *literal.subst* **and** *sub-vars* = *literal.vars* **and**
sub-to-ground = *literal.to-ground* **and** *sub-from-ground* = *literal.from-ground* **and**
map = *image-mset* **and** *to-set* = *set-mset* **and** *to-ground-map* = *image-mset* **and**
from-ground-map = *image-mset* **and** *ground-map* = *image-mset* **and** *to-set-ground* = *set-mset* **and**
to-mset = $\lambda x.\ x$
⟨*proof*⟩

notation *clause.order.less*$_G$ (**infix** $\prec_{cG}$ *50*)
notation *clause.order.less-eq*$_G$ (**infix** $\preceq_{cG}$ *50*)

lemma *obtain-maximal-literal*:
assumes
not-empty: $C \neq \{\#\}$ **and**
grounding: *clause.is-ground* $(C \cdot \gamma)$
obtains l
where *is-maximal* $l\ C$ *is-maximal* $(l \cdot l\ \gamma)$ $(C \cdot \gamma)$
⟨*proof*⟩

lemma *obtain-strictly-maximal-literal*:
assumes
grounding: *clause.is-ground* $(C \cdot \gamma)$ **and**

ground-strictly-maximal: *is-strictly-maximal* l_G $(C \cdot \gamma)$
obtains l **where**
is-strictly-maximal l C $l_G = l \cdot l \gamma$
⟨*proof*⟩

lemma *is-maximal-if-grounding-is-maximal*:
assumes
l-in-C: $l \in\# C$ **and**
C-grounding: *clause.is-ground* $(C \cdot \gamma)$ **and**
l-grounding-is-maximal: *is-maximal* $(l \cdot l \gamma)$ $(C \cdot \gamma)$
shows
is-maximal l C
⟨*proof*⟩

lemma *is-strictly-maximal-if-grounding-is-strictly-maximal*:
assumes
l-in-C: $l \in\# C$ **and**
grounding: *clause.is-ground* $(C \cdot \gamma)$ **and**
grounding-strictly-maximal: *is-strictly-maximal* $(l \cdot l \gamma)$ $(C \cdot \gamma)$
shows
is-strictly-maximal l C
⟨*proof*⟩

lemma *unique-maximal-in-ground-clause*:
assumes
clause.is-ground C
is-maximal l C
is-maximal l' C
shows
$l = l'$
⟨*proof*⟩

lemma *unique-strictly-maximal-in-ground-clause*:
assumes
clause.is-ground C
is-strictly-maximal l C
is-strictly-maximal l' C
shows
$l = l'$
⟨*proof*⟩

thm *literal.order.order.strict-iff-order*

abbreviation *ground-is-maximal* **where**
ground-is-maximal l_G $C_G \equiv$ *is-maximal* (*literal.from-ground* l_G) (*clause.from-ground* C_G)

abbreviation *ground-is-strictly-maximal* **where**

ground-is-strictly-maximal l_G $C_G \equiv$
is-strictly-maximal (*literal.from-ground* l_G) (*clause.from-ground* C_G)

sublocale *ground*: *ground-order-with-equality* **where**
$less_t = (\prec_{tG})$
rewrites
$less_{lG}$*-rewrite* [*simp*]: *multiset-extension.multiset-extension* $(\prec_{tG})$ *mset-lit* $= (\prec_{lG})$
and
$less_{cG}$*-rewrite* [*simp*]: *multiset-extension.multiset-extension* $(\prec_{lG})$ $(\lambda x.\ x) = (\prec_{cG})$
and
is-maximal-rewrite [*simp*]: $\bigwedge l_G\ C_G$. *ground.is-maximal* $l_G\ C_G \longleftrightarrow$ *ground-is-maximal* $l_G\ C_G$ **and**
is-strictly-maximal-rewrite [*simp*]:
$\bigwedge l_G\ C_G$. *ground.is-strictly-maximal* $l_G\ C_G \longleftrightarrow$ *ground-is-strictly-maximal* l_G C_G
⟨*proof*⟩

lemma $less_t$*-*$less_l$:
assumes $t_1 \prec_t t_2$
shows
$less_t$*-*$less_l$*-pos*: $t_1 \approx t_3 \prec_l t_2 \approx t_3$ **and**
$less_t$*-*$less_l$*-neg*: $t_1\ !\approx\ t_3 \prec_l t_2\ !\approx\ t_3$
⟨*proof*⟩

lemma *literal-order-less-if-all-lesseq-ex-less-set*:
assumes
$\forall\, t \in$ *set-uprod* (*atm-of* l). $t \cdot t\ \sigma' \preceq_t t \cdot t\ \sigma$
$\exists\, t \in$ *set-uprod* (*atm-of* l). $t \cdot t\ \sigma' \prec_t t \cdot t\ \sigma$
shows $l \cdot l\ \sigma' \prec_l l \cdot l\ \sigma$
⟨*proof*⟩

lemma $less_c$*-add-mset*:
assumes $l \prec_l l'\ C \preceq_c C'$
shows *add-mset* $l\ C \prec_c$ *add-mset* $l'\ C'$
⟨*proof*⟩

lemmas $less_c$*-add-same* [*simp*] =
multp-add-same[*OF literal.order.asymp literal.order.transp*, *folded* $less_c$*-def*]

end

end
theory *Typed-Functional-Substitution-Example*
imports
Typed-Functional-Substitution
Abstract-Substitution.Functional-Substitution-Example
begin

type-synonym $('f, 'ty)$ *fun-types* $= 'f \Rightarrow 'ty\ list \times 'ty$

Inductive predicate defining well-typed terms.

inductive *welltyped* :: $('f, 'ty)$ *fun-types* $\Rightarrow$ $('v, 'ty)$ *var-types* $\Rightarrow$ $('f,'v)$ *term* $\Rightarrow$ $'ty \Rightarrow$ *bool*
  for $\mathcal{F}$ $\mathcal{V}$ **where**
    Var: $\mathcal{V}\ x = \tau \Longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Var* x) τ
  | *Fun*: $\mathcal{F}\ f = (\tau s, \tau) \Longrightarrow$ *list-all2* (*welltyped* $\mathcal{F}$ $\mathcal{V}$) *ts* $\tau s \Longrightarrow$ *welltyped* $\mathcal{F}$ $\mathcal{V}$ (*Fun* *f ts*) τ

global-interpretation *term*: *base-typing welltyped* $\mathcal{F}$ $\mathcal{V}$
⟨*proof*⟩

global-interpretation *functional-substitution-typing* **where**
  welltyped = *welltyped* $\mathcal{F}$ **and**
  subst = *subst-apply-term* **and** *id-subst* = *Var* **and** *comp-subst* = *subst-compose* **and**
  vars = *vars-term* :: $('f, 'v)$ *term* $\Rightarrow$ $'v$ *set*
  for $\mathcal{F}$ :: $('f, 'ty)$ *fun-types*
  ⟨*proof*⟩

A selection of substitution properties for typed terms.

locale *typed-term-subst-properties* =
  welltyped: *base-typed-subst-stability* **where** *welltyped* = *welltyped* $\mathcal{F}$
  for $\mathcal{F}$ :: $('f, 'ty)$ *fun-types*

global-interpretation *term*: *typed-term-subst-properties* **where**
  subst = *subst-apply-term* **and** *id-subst* = *Var* **and** *comp-subst* = *subst-compose* **and**
  vars = *vars-term* :: $('f, 'v)$ *term* $\Rightarrow$ $'v$ *set* **and** $\mathcal{F} = \mathcal{F}$
  for $\mathcal{F}$ :: $'f \Rightarrow 'ty\ list \times 'ty$
⟨*proof*⟩

find-theorems *name*: *is-welltyped-subst-update*

Examples of generated lemmas and definitions

thm

  term.right-unique
  term.welltyped.subst-stability

  is-welltyped-subst-update
  is-welltyped-on-subset
  is-welltyped-id-subst

term *term.is-welltyped*
term *is-welltyped-on*
term *is-welltyped*

end
theory *Typed-Functional-Substitution-Lifting-Example*
imports
Typed-Functional-Substitution-Lifting
Typed-Functional-Substitution-Example
Abstract-Substitution.Functional-Substitution-Lifting-Example
begin

All property locales have corresponding lifting locales

locale *nonground-typing-lifting* =
is-welltyped: *typed-subst-stability-lifting* **where**
sub-welltyped = *sub-welltyped* **and** *base-welltyped* = *welltyped* $\mathcal{F}$
for $\mathcal{F}$ :: (*'f*, *'ty*) *fun-types* **and** *sub-welltyped* :: (*'v* ⇒ *'ty*) ⇒ *'sub* ⇒ *'ty'* ⇒ *bool*

locale *example-typing-lifting* =
fixes $\mathcal{F}$ :: (*'f*, *'ty*) *fun-types*
begin

sublocale *equation*:
typing-lifting' **where**
sub-welltyped = *welltyped* $\mathcal{F}$ **and**
to-set = *fset*
⟨*proof*⟩

sublocale *equation*:
nonground-typing-lifting **where**
base-vars = *vars-term* **and** *base-subst* = *subst-apply-term* **and** *map* = λ*f*. *map-prod*
f f **and**
to-set = *set-prod* **and** *comp-subst* = *subst-compose* **and** *id-subst* = *Var* **and**
sub-vars = *vars-term* **and** *sub-subst* = *subst-apply-term* **and**
sub-welltyped = *welltyped* $\mathcal{F}$
⟨*proof*⟩

Lifted lemmas and definitions

thm
equation.is-welltyped-def
equation.is-welltyped.subst-stability

term *equation.is-welltyped*

We can lift multiple levels

sublocale *equation-set*:
typing-lifting **where**
sub-welltyped = *equation.is-welltyped.welltyped* $\mathcal{V}$ **and**
to-set = *fset*
⟨*proof*⟩

sublocale *equation-set*:
nonground-typing-lifting **where**

base-vars = *vars-term* **and** *base-subst* = *subst-apply-term* **and** *map* = *fimage*
and
to-set = *fset* **and** *comp-subst* = *subst-compose* **and** *id-subst* = *Var* **and**
sub-vars = *equation-subst.vars* **and** *sub-subst* = *equation-subst.subst* **and**
sub-welltyped = *equation.is-welltyped.welltyped*
⟨*proof*⟩

Lifted lemmas and definitions

thm
equation-set.is-welltyped-def
equation-set.is-welltyped.subst-stability

term *equation-set.is-welltyped*

end

Interpretation with unit as type

global-interpretation *example-typing-lifting* λ-. ([], ())⟨*proof*⟩

end