First Order Clause

Balazs Toth

March 17, 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** · *67*)

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

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

abbreviation (*input*) *GHole* (‹$\Box_G$›) **where**
$\Box_G \equiv \Box$

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

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

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

lemma *gctxt-ident-iff-eq-GHole*[*simp*]: $c\langle t\rangle_G = t \longleftrightarrow c = \Box$
⟨*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*]: $\exists M.\ x \in$ *set-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* $\leq$ *count* M x
obtains M' **where** M = *add-mset* x M'
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *multp-implies-one-step-strong*:
fixes *A B I J K* :: *- multiset*
assumes *transp R* **and** *asymp R* **and** *multp R A B*
defines $J \equiv B - A$ **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* (λX :: *'a multiset* . $X + X$)
⟨*proof*⟩

lemma *multp-image-lesseq-if-all-lesseq*:
assumes
asymp: *asymp R* **and**
transp: *transp R* **and**
all-lesseq: $\forall x \in\# X.\ R^{==}\ (f\ x)\ (g\ x)$
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: $\forall x \in\# X.\ R^{==}\ (f\ x)\ (g\ x)$ **and**
ex-less: $\exists x \in\# X.\ R\ (f\ x)\ (g\ x)$
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 $Upair\ a\ b \in upair\ `\ I \longleftrightarrow (a, b) \in I \land (b, a) \in I$
⟨*proof*⟩

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

definition *mset-uprod* :: $'a\ uprod \Rightarrow\ 'a\ multiset$ **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 *predicate-typed* =
fixes *typed* :: $'expr \Rightarrow 'ty \Rightarrow bool$
assumes *right-unique*: *right-unique typed*
begin

abbreviation *is-typed* **where**
is-typed expr $\equiv \exists\tau.$ *typed expr* τ

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

end

definition *uniform-typed-lifting* **where**
uniform-typed-lifting to-set sub-typed expr $\equiv \exists\tau. \forall$ *sub* $\in$ *to-set expr. sub-typed sub* τ

definition *is-typed-lifting* **where**
is-typed-lifting to-set sub-is-typed expr $\equiv \forall$ *sub* $\in$ *to-set expr. sub-is-typed sub*

locale *typing* =
fixes *is-typed is-welltyped*
assumes *is-typed-if-is-welltyped*:
$\bigwedge$*expr. is-welltyped expr* $\Longrightarrow$ *is-typed expr*

locale *explicit-typing* =
typed: *predicate-typed* **where** *typed* = *typed* +
welltyped: *predicate-typed* **where** *typed* = *welltyped*
for *typed welltyped* :: $'expr \Rightarrow 'ty \Rightarrow bool$ +
assumes *typed-if-welltyped*: $\bigwedge$*expr* τ. *welltyped expr* $\tau \Longrightarrow$ *typed expr* τ
begin

abbreviation *is-typed* **where**
is-typed $\equiv$ *typed.is-typed*

abbreviation *is-welltyped* **where**
is-welltyped $\equiv$ *welltyped.is-typed*

sublocale *typing* **where** *is-typed* = *is-typed* **and** *is-welltyped* = *is-welltyped*
⟨*proof*⟩

lemma *typed-welltyped-same-type*:
assumes *typed expr* τ *welltyped expr* τ'
shows $\tau = \tau'$
⟨*proof*⟩

end

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

abbreviation *is-typed* **where**
is-typed ≡ *uniform-typed-lifting to-set sub-typed*

lemmas *is-typed-def* = *uniform-typed-lifting-def*[*of to-set sub-typed*]

abbreviation *is-welltyped* **where**
is-welltyped ≡ *uniform-typed-lifting to-set sub-welltyped*

lemmas *is-welltyped-def* = *uniform-typed-lifting-def*[*of to-set sub-welltyped*]

sublocale *typing* **where** *is-typed* = *is-typed* **and** *is-welltyped* = *is-welltyped*
⟨*proof*⟩

end

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

abbreviation *is-typed* **where**
is-typed ≡ *is-typed-lifting to-set sub-is-typed*

lemmas *is-typed-def* = *is-typed-lifting-def*[*of to-set sub-is-typed*]

abbreviation *is-welltyped* **where**
is-welltyped ≡ *is-typed-lifting to-set sub-is-welltyped*

lemmas *is-welltyped-def* = *is-typed-lifting-def*[*of to-set sub-is-welltyped*]

sublocale *typing* **where** *is-typed* = *is-typed* **and** *is-welltyped* = *is-welltyped*
⟨*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*
for *to-set* :: *'expr* ⇒ *'sub set* +
fixes *sub-is-typed* :: *'sub* ⇒ *bool*
begin

abbreviation (*input*) *is-typed* **where**
is-typed ≡ *is-typed-lifting to-set sub-is-typed*

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

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

end

locale *natural-magma-typing-lifting* = *typing-lifting* + *natural-magma*
begin

sublocale *is-typed*: *natural-magma-is-typed-lifting* **where** *sub-is-typed* = *sub-is-typed*
⟨*proof*⟩

sublocale *is-welltyped*: *natural-magma-is-typed-lifting* **where** *sub-is-typed* = *sub-is-welltyped*
⟨*proof*⟩

end

locale *natural-magma-with-empty-typing-lifting* =
natural-magma-typing-lifting + *natural-magma-with-empty*
begin

sublocale *is-typed*: *natural-magma-with-empty-is-typed-lifting* **where** *sub-is-typed*
= *sub-is-typed*
⟨*proof*⟩

sublocale *is-welltyped*: *natural-magma-with-empty-is-typed-lifting* **where**
sub-is-typed = *sub-is-welltyped*
⟨*proof*⟩

end

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*
begin

sublocale *natural-magma-with-empty-typing-lifting* **where**
to-set = *set-mset* **and** *plus* = (+) **and** *wrap* = $\lambda l.$ {#*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*: $[\![\mathcal{P} \in \{Pos, Neg\}; \mathcal{P} = Pos \Longrightarrow P; \mathcal{P} = Neg \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *map-literal-inverse*:
$(\bigwedge x.\ f\ (g\ x) = x) \Longrightarrow (\bigwedge l.\ \textit{map-literal}\ f\ (\textit{map-literal}\ g\ l) = l)$
⟨*proof*⟩

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

lemma *literals-distinct* [*simp*]: $Pos \neq Neg$ $Neg \neq Pos$
⟨*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 $(Upair\ t\ t') \in (\lambda(t_1,\ t_2).\ Upair\ t_1\ t_2)$ ‘ $I \longleftrightarrow (t,\ t') \in I$
⟨*proof*⟩

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

abbreviation *Pos-Upair* (**infix** ≈ *66*) **where**
$Pos\text{-}Upair\ t\ t' \equiv Pos\ (Upair\ t\ t')$

abbreviation *Neg-Upair* (**infix** !≈ *66*) **where**
$Neg\text{-}Upair\ t\ t' \equiv Neg\ (Upair\ t\ t')$

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*]: $\exists \iota.\ f \in$ *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: *explicit-typing term-typed term-welltyped*
  for *term-typed term-welltyped*
begin

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

lemma *atom-is-typed-iff* [*simp*]:
  $atom.is\text{-}typed\ (Upair\ t\ t') \longleftrightarrow (\exists \tau.\ term\text{-}typed\ t\ \tau \land term\text{-}typed\ t'\ \tau)$
  ⟨*proof*⟩

lemma *atom-is-welltyped-iff* [*simp*]:
  $atom.is\text{-}welltyped\ (Upair\ t\ t') \longleftrightarrow (\exists \tau.\ term\text{-}welltyped\ t\ \tau \land term\text{-}welltyped\ t'\ \tau)$
  ⟨*proof*⟩

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

lemma *literal-is-typed-iff* [*simp*]:
  $literal.is\text{-}typed\ (t \approx t') \longleftrightarrow atom.is\text{-}typed\ (Upair\ t\ t')$
  $literal.is\text{-}typed\ (t\ !\!\approx t') \longleftrightarrow atom.is\text{-}typed\ (Upair\ t\ t')$
  ⟨*proof*⟩

lemma *literal-is-welltyped-iff* [*simp*]:
  $literal.is\text{-}welltyped\ (t \approx t') \longleftrightarrow atom.is\text{-}welltyped\ (Upair\ t\ t')$
  $literal.is\text{-}welltyped\ (t\ !\!\approx t') \longleftrightarrow atom.is\text{-}welltyped\ (Upair\ t\ t')$
  ⟨*proof*⟩

lemma *literal-is-typed-iff-atm-of*: $literal.is\text{-}typed\ l \longleftrightarrow atom.is\text{-}typed\ (atm\text{-}of\ l)$
  ⟨*proof*⟩

lemma *literal-is-welltyped-iff-atm-of*:
  $literal.is\text{-}welltyped\ l \longleftrightarrow atom.is\text{-}welltyped\ (atm\text{-}of\ l)$
  ⟨*proof*⟩

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

end

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

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

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

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

locale *context-compatible-typing* =
  fixes *Fun typed*
  assumes
   context-compatible [*intro*]:
    $\bigwedge t\ t'\ c\ \tau\ \tau'.$
      typed $t\ \tau' \Longrightarrow$
      typed $t'\ \tau' \Longrightarrow$
      typed (*Fun*$\langle c;\ t\rangle$) $\tau \Longrightarrow$
      typed (*Fun*$\langle c;\ t'\rangle$) τ

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

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

end

locale *term-typing* =
  explicit-typing +
  typed: *context-compatible-typing* **where** *typed* = *typed* +
  welltyped: *context-compatible-typing* **where** *typed* = *welltyped* +
  welltyped: *subterm-typing* **where** *typed* = *welltyped* +
assumes *all-terms-are-typed*: $\bigwedge t.$ *is-typed t*
begin

sublocale *typed*: *subterm-typing*
  ⟨*proof*⟩

end

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

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

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

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

abbreviation *typed* **where** *typed* ≡ *Ground-Typing.typed* $\mathcal{F}$
abbreviation *welltyped* **where** *welltyped* ≡ *Ground-Typing.welltyped* $\mathcal{F}$

sublocale *explicit-typing* **where** *typed* = *typed* **and** *welltyped* = *welltyped*
⟨*proof*⟩

sublocale *term-typing* **where** *typed* = *typed* **and** *welltyped* = *welltyped* **and** *Fun* = *GFun*
⟨*proof*⟩

end

locale *ground-typing* = *term*: *ground-term-typing*
begin

sublocale *clause-typing* **where** *term-typed* = *term.typed* **and** *term-welltyped* = *term.welltyped*
  ⟨*proof*⟩

end

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** $\cdot$ *67*)
notation *subst-apply-term* (**infixl** $\cdot t$ *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 = (⊙) **and** *id-subst* = *Var* **and** *base-subst* = (·*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 = (·*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
$\bigwedge c\ \sigma.$ *subst c σ* = $c \cdot t_c\ \sigma$ **and**
$\bigwedge c.$ *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* $c\langle t\rangle = (\textit{to-ground } c)\langle \textit{term.to-ground } t\rangle_G$
⟨*proof*⟩

lemma *term-from-ground-context-from-ground* [*simp*]:
term.from-ground $c_G\langle t_G\rangle_G = (\textit{from-ground } c_G)\langle \textit{term.from-ground } t_G\rangle$
⟨*proof*⟩

lemma *term-from-ground-context-to-ground*:
assumes *is-ground c*
shows *term.from-ground* $(\textit{to-ground } c)\langle t_G\rangle_G = c\langle \textit{term.from-ground } t_G\rangle$
⟨*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 $(c \circ_c c') \longleftrightarrow$ *is-ground c* $\wedge$ *is-ground* c'
⟨*proof*⟩

lemma *ground-context-subst*:
assumes
is-ground c_G
$c_G = (c \cdot t_c\ \sigma) \circ_c c'$
shows
$c_G = c \circ_c c' \cdot t_c\ \sigma$
⟨*proof*⟩

lemma *from-ground-hole* [*simp*]: *from-ground* $c_G = \Box \longleftrightarrow c_G = \Box$
⟨*proof*⟩

lemma *hole-simps* [*simp*]: *from-ground* $\Box = \Box$ *to-ground* $\Box = \Box$
⟨*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* = λ*l*. {#*l*#} **and** *plus-ground* = (+) **and** *wrap-ground* = λ*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* = ($\odot$)
⟨*proof*⟩

end

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

3.1 Nonground Atoms

sublocale *atom*: *term-based-lifting* **where**
sub-subst = ($\cdot t$) **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 $\cup$ *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' = $a \cdot a\ \sigma$
obtains t_1 t_2
where a = *Upair* t_1 t_2 $t_1' = t_1 \cdot t\ \sigma$ $t_2' = t_2 \cdot t\ \sigma$
⟨*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** ·*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 $a \cdot l\ \sigma$ = *Pos* ($a \cdot a\ \sigma$)
Neg $a \cdot l\ \sigma$ = *Neg* ($a \cdot a\ \sigma$)
atm-of ($l \cdot l\ \sigma$) = *atm-of* $l \cdot a\ \sigma$
⟨*proof*⟩

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

lemma *subst-polarity-stable*:
shows
subst-neg-stable [*simp*]: *is-neg* ($l \cdot l\ \sigma$) ⟷ *is-neg* l **and**
subst-pos-stable [*simp*]: *is-pos* ($l \cdot l\ \sigma$) ⟷ *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 $l \cdot l\ \sigma = t_1' \approx t_2'$
obtains $t_1\ t_2$
where $l = t_1 \approx t_2\ t_1' = t_1 \cdot t\ \sigma\ t_2' = t_2 \cdot t\ \sigma$
⟨*proof*⟩

lemma *obtain-from-neg-literal-subst*:
assumes $l \cdot l\ \sigma = t_1'\ !\!\approx t_2'$
obtains $t_1\ t_2$
where $l = t_1\ !\!\approx t_2\ t_1 \cdot t\ \sigma = t_1'\ t_2 \cdot t\ \sigma = t_2'$
⟨*proof*⟩

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

3.3 Nonground Literals - Alternative

lemma *uprod-literal* [*simp*]:
fixes *l*
shows
functional-substitution-lifting.subst (·*t*) *map-uprod-literal l* σ = *l* ·*l* σ
functional-substitution-lifting.vars term.vars uprod-literal-to-set l = *literal.vars l*
grounding-lifting.from-ground term.from-ground map-uprod-literal l_G = *literal.from-ground* l_G
grounding-lifting.to-ground term.to-ground map-uprod-literal l = *literal.to-ground* *l*
⟨*proof*⟩

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
$\bigwedge l\ \sigma$. *uprod-literal.subst* $l\ \sigma$ = *literal.subst* $l\ \sigma$ **and**
$\bigwedge l$. *uprod-literal.vars* l = *literal.vars* l **and**
$\bigwedge l_G$. *uprod-literal.from-ground* l_G = *literal.from-ground* l_G **and**
$\bigwedge l$. *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** · *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* ((*select* C) $\cdot$ γ)

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 ∈# *select* (*clause.from-ground* C)
shows *literal.is-ground* l
⟨*proof*⟩

lemma *ground-literal-in-selection*:
assumes *clause.is-ground* C l_G ∈# *clause.to-ground* C
shows *literal.from-ground* l_G ∈# 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 ∈# *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 *Infinite-Variables-Per-Type*
imports
HOL−Library.Countable-Set
HOL−Cardinals.Cardinals
Fresh-Identifiers.Fresh
begin

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

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

definition *infinite-variables-per-type* :: $('v \Rightarrow 'ty) \Rightarrow bool$ **where**
infinite-variables-per-type $\mathcal{V} \equiv \forall\, ty.\ infinite\ \{x.\ \mathcal{V}\ x = ty\}$

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
$X \cap f\ `\ Y = \{\}$
$\forall\, x \in Y.\ \mathcal{V}\ (f\ x) = \mathcal{V}\ x$
⟨*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'
$f\ `\ X \cap f'\ `\ Y = \{\}$
$\forall\, x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
$\forall\, x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
⟨*proof*⟩

lemma *obtain-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 ' $X \cap f'$ ' $Y = \{\}$
$\forall x \in X.\ \mathcal{V}_1\ (f\ x) = \mathcal{V}_1\ x$
$\forall x \in Y.\ \mathcal{V}_2\ (f'\ x) = \mathcal{V}_2\ x$
⟨*proof*⟩

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

lemma *obtain-infinite-variables-per-type*:
assumes $|UNIV :: 'ty\ set| \leq o\ |UNIV :: 'v\ set|$
obtains $\mathcal{V} :: 'v :: infinite \Rightarrow 'ty$ **where** *infinite-variables-per-type* $\mathcal{V}$
⟨*proof*⟩

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 *Typed-Functional-Substitution*
imports
Typing
Abstract-Substitution.Functional-Substitution
Infinite-Variables-Per-Type
Collect-Extra
begin

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

locale *explicitly-typed-functional-substitution* =
base-functional-substitution **where** *vars* = *vars* **and** *id-subst* = *id-subst*
for
id-subst :: $'var \Rightarrow 'base$ **and**
vars :: $'base \Rightarrow 'var\ set$ **and**
typed :: $('var, 'ty)$ *var-types* $\Rightarrow 'base \Rightarrow 'ty \Rightarrow bool$ +

assumes
  predicate-typed: $\bigwedge\mathcal{V}$. *predicate-typed* (*typed* $\mathcal{V}$) **and**
  typed-id-subst [*intro*]: $\bigwedge\mathcal{V}$ *x*. *typed* $\mathcal{V}$ (*id-subst x*) ($\mathcal{V}$ *x*)
begin

sublocale *predicate-typed typed* $\mathcal{V}$
  ⟨*proof*⟩

abbreviation *is-typed-on* :: *'var set* ⇒ (*'var*, *'ty*) *var-types* ⇒ (*'var* ⇒ *'base*) ⇒ *bool* **where**
  $\bigwedge\mathcal{V}$. *is-typed-on X* $\mathcal{V}$ σ ≡ ∀ *x* ∈ *X*. *typed* $\mathcal{V}$ (σ *x*) ($\mathcal{V}$ *x*)

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

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

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

end

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

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

abbreviation *is-typed-ground-instance* **where**
  is-typed-ground-instance expr $\mathcal{V}$ γ ≡
    is-ground (*expr* · γ) ∧
    is-typed $\mathcal{V}$ *expr* ∧
    base.is-typed-on (*vars expr*) $\mathcal{V}$ γ ∧
    infinite-variables-per-type $\mathcal{V}$

end

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

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

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

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

end

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

sublocale *typed-grounding-functional-substitution* **where**
base-subst = *subst* **and** *base-vars* = *vars* **and** *is-typed* = *is-typed* **and**
base-typed = *typed*
⟨*proof*⟩

end

locale *inhabited-typed-functional-substitution* =
typed-functional-substitution +
base: *inhabited-explicitly-typed-functional-substitution* **where**
subst = *base-subst* **and** *vars* = *base-vars* **and** *typed* = *base-typed*
begin

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

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

end

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

locale *typed-subst-stability* = *typed-functional-substitution* +
assumes
  subst-stability [*simp*]:
    $\bigwedge\mathcal{V}\ expr\ \sigma.\ base.is\text{-}typed\text{-}on\ (vars\ expr)\ \mathcal{V}\ \sigma \Longrightarrow is\text{-}typed\ \mathcal{V}\ (expr \cdot \sigma) \longleftrightarrow$
$is\text{-}typed\ \mathcal{V}\ expr$
begin

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

end

locale *explicitly-typed-subst-stability* = *explicitly-typed-functional-substitution* +
assumes
  explicit-subst-stability [*simp*]:
    $\bigwedge\mathcal{V}\ expr\ \sigma\ \tau.\ is\text{-}typed\text{-}on\ (vars\ expr)\ \mathcal{V}\ \sigma \Longrightarrow typed\ \mathcal{V}\ (expr \cdot \sigma)\ \tau \longleftrightarrow typed$
$\mathcal{V}\ expr\ \tau$
begin

lemma *explicit-subst-stability-UNIV* [*simp*]:
  $\bigwedge\mathcal{V}\ expr\ \sigma.\ is\text{-}typed\text{-}on\ UNIV\ \mathcal{V}\ \sigma \Longrightarrow typed\ \mathcal{V}\ (expr \cdot \sigma)\ \tau \longleftrightarrow typed\ \mathcal{V}\ expr\ \tau$
  ⟨*proof*⟩

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

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

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

end

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

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

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

end

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

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

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

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

end

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

locale *explicitly-typed-renaming* =
explicitly-typed-functional-substitution **where** *typed* = *typed* +
renaming-variables +
explicitly-replaceable-$\mathcal{V}$ **where** *typed* = *typed*
for *typed* :: ('*var* ⇒ '*ty*) ⇒ '*expr* ⇒ '*ty* ⇒ *bool* +
assumes
explicit-typed-renaming [*simp*]:
$\bigwedge \mathcal{V}\ \mathcal{V}'\ expr\ \varrho\ \tau.\ is\text{-}renaming\ \varrho \Longrightarrow$
$\forall x \in vars\ expr.\ \mathcal{V}\ x = \mathcal{V}'\ (rename\ \varrho\ x) \Longrightarrow$
$typed\ \mathcal{V}'\ (expr \cdot \varrho)\ \tau \longleftrightarrow typed\ \mathcal{V}\ expr\ \tau$
begin

sublocale *typed-renaming*
where *base-vars* = *vars* **and** *base-subst* = *subst* **and** *base-typed* = *typed* **and** *is-typed* = *is-typed*
⟨*proof*⟩

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

lemma *inj-id-subst*: *inj id-subst*
⟨*proof*⟩

lemma *obtain-typed-renaming*:
fixes $\mathcal{V}$ :: '*var* ⇒ '*ty*
assumes
finite X

infinite-variables-per-type $\mathcal{V}$
obtains ϱ :: *'var* ⇒ *'expr* **where**
is-renaming ϱ
id-subst ' $X \cap \varrho$ ' $Y = \{\}$
is-typed-on $Y\ \mathcal{V}\ \varrho$
⟨*proof*⟩

lemma *obtain-typed-renamings*:
fixes $\mathcal{V}_1\ \mathcal{V}_2$ :: *'var* ⇒ *'ty*
assumes
finite X
infinite-variables-per-type $\mathcal{V}_2$
obtains $\varrho_1\ \varrho_2$ :: *'var* ⇒ *'expr* **where**
is-renaming ϱ_1
is-renaming ϱ_2
ϱ_1 ' $X \cap \varrho_2$ ' $Y = \{\}$
is-typed-on $X\ \mathcal{V}_1\ \varrho_1$
is-typed-on $Y\ \mathcal{V}_2\ \varrho_2$
⟨*proof*⟩

lemma *obtain-typed-renamings'*:
fixes $\mathcal{V}_1\ \mathcal{V}_2$ :: *'var* ⇒ *'ty*
assumes
finite Y
infinite-variables-per-type $\mathcal{V}_1$
obtains $\varrho_1\ \varrho_2$ :: *'var* ⇒ *'expr* **where**
is-renaming ϱ_1
is-renaming ϱ_2
ϱ_1 ' $X \cap \varrho_2$ ' $Y = \{\}$
is-typed-on $X\ \mathcal{V}_1\ \varrho_1$
is-typed-on $Y\ \mathcal{V}_2\ \varrho_2$
⟨*proof*⟩

lemma *renaming-subst-compose*:
assumes
is-renaming ϱ
is-typed-on $X\ \mathcal{V}\ (\varrho \odot \sigma)$
is-typed-on $X\ \mathcal{V}\ \varrho$
shows *is-typed-on* ($\bigcup$ (*vars* ' ϱ ' X)) $\mathcal{V}\ \sigma$
⟨*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**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**

finite-vars: *finite* (*vars expr*)
obtains $\mathcal{V}_3$ **where**
∀ *x*∈*vars expr*. $\mathcal{V}_1$ *x* = $\mathcal{V}_3$ (*rename* ϱ_1 *x*)
∀ *x*∈*vars expr′*. $\mathcal{V}_2$ *x* = $\mathcal{V}_3$ (*rename* ϱ_2 *x*)
infinite-variables-per-type $\mathcal{V}_3$
⟨*proof*⟩

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* · ϱ_1) ∩ *vars* (*expr′* · ϱ_2) = {} **and**
$\mathcal{V}_1$: *infinite-variables-per-type* $\mathcal{V}_1$ **and**
finite-vars: *finite* (*vars expr′*)
obtains $\mathcal{V}_3$ **where**
∀ *x*∈*vars expr*. $\mathcal{V}_1$ *x* = $\mathcal{V}_3$ (*rename* ϱ_1 *x*)
∀ *x*∈*vars expr′*. $\mathcal{V}_2$ *x* = $\mathcal{V}_3$ (*rename* ϱ_2 *x*)
infinite-variables-per-type $\mathcal{V}_3$
⟨*proof*⟩

locale *based-typed-renaming* =
base: *explicitly-typed-renaming* **where**
subst = *base-subst* **and** *vars* = *base-vars* :: *′base* ⇒ *′v set* **and**
typed = *typed* :: (*′v* ⇒ *′ty*) ⇒ *′base* ⇒ *′ty* ⇒ *bool* +
base: *explicitly-typed-functional-substitution* **where**
vars = *base-vars* **and** *subst* = *base-subst* +
based-functional-substitution +
renaming-variables
begin

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

lemma *obtain-merged-grounding*:
fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: *′v* ⇒ *′ty*
assumes
base.is-typed-on (*vars expr*) $\mathcal{V}_1$ γ_1
base.is-typed-on (*vars expr′*) $\mathcal{V}_2$ γ_2
is-ground (*expr* · γ_1)
is-ground (*expr′* · γ_2) **and**
$\mathcal{V}_2$: *infinite-variables-per-type* $\mathcal{V}_2$ **and**
finite-vars: *finite* (*vars expr*)
obtains ϱ_1 ϱ_2 γ **where**

base.is-renaming ϱ_1
base.is-renaming ϱ_2
vars $(expr \cdot \varrho_1) \cap$ *vars* $(expr' \cdot \varrho_2) = \{\}$
base.is-typed-on (*vars expr*) $\mathcal{V}_1$ ϱ_1
base.is-typed-on (*vars expr'*) $\mathcal{V}_2$ ϱ_2
$\forall X \subseteq$ *vars expr*. $\forall x \in X.\ \gamma_1\ x = (\varrho_1 \odot \gamma)\ x$
$\forall X \subseteq$ *vars expr'*. $\forall x \in X.\ \gamma_2\ x = (\varrho_2 \odot \gamma)\ x$
⟨*proof*⟩

lemma *obtain-merged-grounding'*:
fixes $\mathcal{V}_1$ $\mathcal{V}_2$ :: *'v* ⇒ *'ty*
assumes
typed-γ_1: *base.is-typed-on* (*vars expr*) $\mathcal{V}_1$ γ_1 **and**
typed-γ_2: *base.is-typed-on* (*vars expr'*) $\mathcal{V}_2$ γ_2 **and**
expr-grounding: *is-ground* $(expr \cdot \gamma_1)$ **and**
expr'-grounding: *is-ground* $(expr' \cdot \gamma_2)$ **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 $(expr \cdot \varrho_1) \cap$ *vars* $(expr' \cdot \varrho_2) = \{\}$
base.is-typed-on (*vars expr*) $\mathcal{V}_1$ ϱ_1
base.is-typed-on (*vars expr'*) $\mathcal{V}_2$ ϱ_2
$\forall X \subseteq$ *vars expr*. $\forall x \in X.\ \gamma_1\ x = (\varrho_1 \odot \gamma)\ x$
$\forall X \subseteq$ *vars expr'*. $\forall x \in X.\ \gamma_2\ x = (\varrho_2 \odot \gamma)\ x$
⟨*proof*⟩

end

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

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

locale *subst-is-typed-abbreviations* =
is-typed: *typed-functional-substitution* **where**
base-typed = *base-typed* **and** *is-typed* = *expr-is-typed* +
is-welltyped: *typed-functional-substitution* **where**
base-typed = *base-welltyped* **and** *is-typed* = *expr-is-welltyped*
for
base-typed base-welltyped :: (*'var*, *'ty*) *var-types* ⇒ *'base* ⇒ *'ty* ⇒ *bool* **and**
expr-is-typed expr-is-welltyped :: (*'var*, *'ty*) *var-types* ⇒ *'expr* ⇒ *bool*
begin

abbreviation *is-typed-on* **where**
is-typed-on ≡ *is-typed.base.is-typed-on*

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

abbreviation *is-typed* **where**
is-typed ≡ *is-typed.base.is-typed-on UNIV*

abbreviation *is-welltyped* **where**
is-welltyped ≡ *is-welltyped.base.is-typed-on UNIV*

end

locale *functional-substitution-typing* =
is-typed: *typed-functional-substitution* **where**
base-typed = *base-typed* **and** *is-typed* = *is-typed* +
is-welltyped: *typed-functional-substitution* **where**
base-typed = *base-welltyped* **and** *is-typed* = *is-welltyped*
for
base-typed base-welltyped :: (*'var*, *'ty*) *var-types* ⇒ *'base* ⇒ *'ty* ⇒ *bool* **and**
is-typed is-welltyped :: (*'var*, *'ty*) *var-types* ⇒ *'expr* ⇒ *bool* +
assumes *typing*: ⋀$\mathcal{V}$. *typing* (*is-typed* $\mathcal{V}$) (*is-welltyped* $\mathcal{V}$)
begin

sublocale *base*: *typing is-typed* $\mathcal{V}$ *is-welltyped* $\mathcal{V}$
⟨*proof*⟩

sublocale *subst*: *subst-is-typed-abbreviations*
where *expr-is-typed* = *is-typed* **and** *expr-is-welltyped* = *is-welltyped*
⟨*proof*⟩

end

locale *base-functional-substitution-typing* =
typed: *explicitly-typed-functional-substitution* **where** *typed* = *typed* +
welltyped: *explicitly-typed-functional-substitution* **where** *typed* = *welltyped*
for
welltyped typed :: (*'var*, *'ty*) *var-types* ⇒ *'expr* ⇒ *'ty* ⇒ *bool* +
assumes
explicit-typing: ⋀$\mathcal{V}$. *explicit-typing* (*typed* $\mathcal{V}$) (*welltyped* $\mathcal{V}$)
begin

sublocale *base*: *explicit-typing typed* $\mathcal{V}$ *welltyped* $\mathcal{V}$
⟨*proof*⟩

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

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

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

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

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

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

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

sublocale *subst*: *typing subst.is-typed-on X* $\mathcal{V}$ *subst.is-welltyped-on X* $\mathcal{V}$
⟨*proof*⟩

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

abbreviation (*input*) *lifted-is-typed* **where**
lifted-is-typed $\mathcal{V}$ ≡ *is-typed-lifting to-set* (*sub-is-typed* $\mathcal{V}$)

lemmas *lifted-is-typed-def* = *is-typed-lifting-def*[*of to-set*, *THEN ext-equiv*, *of sub-is-typed*]

sublocale *typed-functional-substitution* **where**

vars = *vars* **and** *subst* = *subst* **and** *is-typed* = *lifted-is-typed*
⟨*proof*⟩

end

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

abbreviation (*input*) *lifted-is-typed* **where**
lifted-is-typed $\mathcal{V}$ ≡ *uniform-typed-lifting to-set* (*base-typed* $\mathcal{V}$)

lemmas *lifted-is-typed-def* = *uniform-typed-lifting-def*[*of to-set*, *THEN ext-equiv*, *of base-typed*]

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

end

locale *uniform-typed-grounding-functional-substitution-lifting* =
uniform-typed-functional-substitution-lifting +
grounding-lifting **where** *sub-subst* = *base-subst* **and** *sub-vars* = *base-vars* +
base: *explicitly-typed-grounding-functional-substitution* **where**
vars = *base-vars* **and** *subst* = *base-subst* **and** *typed* = *base-typed* **and**
to-ground = *sub-to-ground* **and** *from-ground* = *sub-from-ground*
begin

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

end

locale *typed-grounding-functional-substitution-lifting* =
typed-functional-substitution-lifting +
grounding-lifting +
sub: *typed-grounding-functional-substitution* **where**
vars = *sub-vars* **and** *subst* = *sub-subst* **and** *is-typed* = *sub-is-typed* **and**
to-ground = *sub-to-ground* **and** *from-ground* = *sub-from-ground*

begin

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

end

locale *uniform-inhabited-typed-functional-substitution-lifting* =
uniform-typed-functional-substitution-lifting +
base: *inhabited-explicitly-typed-functional-substitution* **where**
vars = *base-vars* **and** *subst* = *base-subst* **and** *typed* = *base-typed*
begin

sublocale *inhabited-typed-functional-substitution* **where**
vars = *vars* **and** *subst* = *subst* **and** *is-typed* = *lifted-is-typed*
⟨*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** *is-typed* = *sub-is-typed*
begin

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

end

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

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

end

locale *uniform-typed-subst-stability-lifting* =
uniform-typed-functional-substitution-lifting +
base: *explicitly-typed-subst-stability* **where**

typed = *base-typed* **and** *vars* = *base-vars* **and** *subst* = *base-subst*
begin

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

end

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

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

end

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

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

end

locale *based-typed-renaming-lifting* =
based-functional-substitution-lifting +
renaming-variables-lifting +
based-typed-renaming **where** *subst* = *sub-subst* **and** *vars* = *sub-vars*
begin

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

end

locale *typed-renaming-lifting* =
typed-functional-substitution-lifting **where**
base-typed = *base-typed* :: $('v \Rightarrow 'ty) \Rightarrow 'base \Rightarrow 'ty \Rightarrow bool$ +
based-typed-renaming-lifting **where** *typed* = *base-typed* +
sub: *typed-renaming* **where**

subst = *sub-subst* **and** *vars* = *sub-vars* **and** *is-typed* = *sub-is-typed*
begin

sublocale *typed-renaming* **where**
subst = *subst* **and** *vars* = *vars* **and** *is-typed* = *lifted-is-typed*
⟨*proof*⟩

end

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

sublocale *typed-renaming* **where**
is-typed = *lifted-is-typed* **and** *subst* = *subst* **and** *vars* = *vars*
⟨*proof*⟩

end

end
theory *Functional-Substitution-Typing-Lifting*
imports
Functional-Substitution-Typing
Typed-Functional-Substitution-Lifting
begin

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

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

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

end

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

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

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

end

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

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

locale *base-typing-properties* =
base-functional-substitution-typing +
typed: *base-typed-properties* +
welltyped: *base-typed-properties* **where** *typed* = *welltyped*

locale *base-inhabited-typing-properties* =
base-typing-properties +
typed: *inhabited-explicitly-typed-functional-substitution* +
welltyped: *inhabited-explicitly-typed-functional-substitution* **where** *typed* = *welltyped*

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

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

Note: Implicitly implies that every function symbol has a fixed arity

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* = (τ*s*, τ) ⟹ *list-all2* (*welltyped* $\mathcal{V}$) *ts* τ*s* ⟹ *welltyped* $\mathcal{V}$ (*Fun f ts*) τ

sublocale *term*: *explicit-typing typed* ($\mathcal{V}$ :: (*'v*, *'ty*) *var-types*) *welltyped* $\mathcal{V}$
⟨*proof*⟩

sublocale *term*: *term-typing* **where**
  typed = *typed* ($\mathcal{V}$ :: *'v* ⇒ *'ty*) **and** *welltyped* = *welltyped* $\mathcal{V}$ **and** *Fun* = *Fun*
⟨*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** *typed* = *typed* **and** *to-ground* = *term.to-ground* **and**
  from-ground = *term.from-ground*
⟨*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*: ⋀τ. ∃*f*. $\mathcal{F}$ *f* = ([], τ)
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** *typed* = *typed* **and** *to-ground* = *term.to-ground* **and**
  from-ground = *term.from-ground*
⟨*proof*⟩

end

end
theory *Nonground-Typing*

imports
Clause-Typing
Functional-Substitution-Typing-Lifting
Nonground-Term-Typing
Nonground-Clause
begin

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

locale *nonground-uniform-typed-lifting* =
uniform-typed-subst-stability-lifting +
uniform-replaceable-𝒱-lifting +
uniform-typed-renaming-lifting +
uniform-typed-grounding-functional-substitution-lifting

locale *nonground-typed-lifting* =
typed-subst-stability-lifting +
replaceable-𝒱-lifting +
typed-renaming-lifting +
typed-grounding-functional-substitution-lifting

locale *nonground-uniform-typing-lifting* =
functional-substitution-uniform-typing-lifting +
is-typed: *nonground-uniform-typed-lifting* **where** *base-typed* = *base-typed* +
is-welltyped: *nonground-uniform-typed-lifting* **where** *base-typed* = *base-welltyped*
begin

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

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

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

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

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

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

end

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

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

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

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

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

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

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

end

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

locale *nonground-inhabited-typing-lifting* =
nonground-typing-lifting +
is-typed: *inhabited-typed-functional-substitution-lifting* **where** *base-typed* = *base-typed* +
is-welltyped: *inhabited-typed-functional-substitution-lifting* **where**
sub-is-typed = *sub-is-welltyped* **and** *base-typed* = *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 *term-based-nonground-uniform-typing-lifting* =
term: *nonground-term* +
nonground-uniform-typing-lifting **where**
id-subst = *Var* **and** *comp-subst* = (⊙) **and** *map* = *map-uprod* **and** *to-set* = *set-uprod* **and**
base-vars = *term.vars* **and** *base-subst* = (·*t*) **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*

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

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

sublocale *clause-typing typed* ($\mathcal{V}$ :: ('*v*, '*ty*) *var-types*) *welltyped* $\mathcal{V}$
⟨*proof*⟩

sublocale *atom*: *term-based-nonground-uniform-typing-lifting* **where**
base-typed = *typed* :: ('*v* ⇒ '*ty*) ⇒ ('*f*, '*v*) *Term.term* ⇒ '*ty* ⇒ *bool* **and**
base-welltyped = *welltyped*
⟨*proof*⟩

sublocale *literal*: *term-based-nonground-typing-lifting* **where**
base-typed = *typed* :: ('*v* ⇒ '*ty*) ⇒ ('*f*, '*v*) *Term.term* ⇒ '*ty* ⇒ *bool* **and**
base-welltyped = *welltyped* **and** *sub-vars* = *atom.vars* **and** *sub-subst* = (·*a*) **and**
map = *map-literal* **and** *to-set* = *set-literal* **and** *sub-is-typed* = *atom.is-typed* **and**
sub-is-welltyped = *atom.is-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-typed = *typed* **and** *base-welltyped* = *welltyped* **and**
sub-vars = *literal.vars* **and** *sub-subst* = (·*l*) **and** *map* = *image-mset* **and** *to-set* = *set-mset* **and**
sub-is-typed = *literal.is-typed* **and** *sub-is-welltyped* = *literal.is-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-uniform-inhabited-typing-lifting* **where**
base-typed = *typed* :: (*'v* ⇒ *'ty*) ⇒ (*'f*, *'v*) *Term.term* ⇒ *'ty* ⇒ *bool* **and**
base-welltyped = *welltyped*
⟨*proof*⟩

sublocale *literal*: *term-based-nonground-inhabited-typing-lifting* **where**
base-typed = *typed* :: (*'v* ⇒ *'ty*) ⇒ (*'f*, *'v*) *Term.term* ⇒ *'ty* ⇒ *bool* **and**
base-welltyped = *welltyped* **and** *sub-vars* = *atom.vars* **and** *sub-subst* = (·*a*) **and**
map = *map-literal* **and** *to-set* = *set-literal* **and** *sub-is-typed* = *atom.is-typed* **and**
sub-is-welltyped = *atom.is-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-typed = *typed* **and** *base-welltyped* = *welltyped* **and**
sub-vars = *literal.vars* **and** *sub-subst* = (·*l*) **and** *map* = *image-mset* **and** *to-set*
= *set-mset* **and**
sub-is-typed = *literal.is-typed* **and** *sub-is-welltyped* = *literal.is-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* ⟹ *P x2 y2* ⟹ (*x1*, *y1*) = (*x2*, *y2*)
shows $\exists_{\leq 1}$(*x*, *y*). *P x y*
⟨*proof*⟩

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

⟨*proof*⟩

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

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

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

lemma *Collect-eq-if-Uniq-prod*:
$(\exists_{\leq 1}(x,\ y).\ P\ x\ y) \Longrightarrow \{(x,\ y).\ P\ x\ y\} = \{\} \lor (\exists\, x\ y.\ \{(x,\ y).\ P\ x\ y\} = \{(x,\ y)\})$
⟨*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 $select_G\ C_G\ C\ \mathcal{V}\ \gamma \equiv$
$C \cdot \gamma = clause.from\text{-}ground\ C_G \land$
$select_G\ C_G = clause.to\text{-}ground\ ((select\ C) \cdot \gamma) \land$
clause.is-welltyped-ground-instance $C\ \mathcal{V}\ \gamma$

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

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* ⇒ ('*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 ≡ *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 $I \longleftrightarrow (\forall t\ t'\ ctxt.\ (t, t') \in I \longrightarrow (ctxt\langle t\rangle_G, ctxt\langle t'\rangle_G) \in I)$

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

lemma *compatible-with-gctxt-converse*:

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

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

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

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

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

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

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

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

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

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

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

lemma *rewrite-inside-gctxt-insert*:

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

lemma *converse-rewrite-steps*: (*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*) ∈ *rewrite-inside-gctxt R* **and**
ball-R-rhs-lt-lhs: ⋀*t1 t2*. (*t1*, *t2*) ∈ *R* ⟹ *t2* $\prec_t$ *t1* **and**
compatible-with-gctxt: ⋀*t1 t2 ctxt*. *t2* $\prec_t$ *t1* ⟹ *ctxt*⟨*t2*⟩$_G$ $\prec_t$ *ctxt*⟨*t1*⟩$_G$
shows *t2* $\prec_t$ *t1*
⟨*proof*⟩

lemma *mem-rewrite-step-union-NF*:
assumes (*t*, *t'*) ∈ *rewrite-inside-gctxt* (*R1* ∪ *R2*)
t ∈ *NF* (*rewrite-inside-gctxt R2*)
shows (*t*, *t'*) ∈ *rewrite-inside-gctxt R1*
⟨*proof*⟩

lemma *predicate-holds-of-mem-rewrite-inside-gctxt*:
assumes *rule-in*: (*t1*, *t2*) ∈ *rewrite-inside-gctxt R* **and**
ball-P: ⋀*t1 t2*. (*t1*, *t2*) ∈ *R* ⟹ *P t1 t2* **and**
preservation: ⋀*t1 t2 ctxt* σ. (*t1*, *t2*) ∈ *R* ⟹ *P t1 t2* ⟹ *P ctxt*⟨*t1*⟩$_G$ *ctxt*⟨*t2*⟩$_G$
shows *P t1 t2*
⟨*proof*⟩

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

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

lemma *wf-converse-rewrite-inside-gctxt*:
fixes *E* :: *'f gterm rel*
assumes
wfP-R: *wfP R* **and**
R-compatible-with-gctxt: ⋀*ctxt t t'*. *R t t'* ⟹ *R ctxt*⟨*t*⟩$_G$ *ctxt*⟨*t'*⟩$_G$ **and**
equations-subset-R: ⋀*x y*. (*x*, *y*) ∈ *E* ⟹ *R y x*
shows *wf* ((*rewrite-inside-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 (γ *var*) $\Longrightarrow$
(*base-to-ground* (γ *var*), *base-to-ground update*) $\in I \Longrightarrow$
based.is-ground (*subst expr* γ) $\Longrightarrow$
entails-def (*subst expr* (γ(*var* := *update*))) $\Longrightarrow$
entails-def (*subst expr* γ)
begin

abbreviation *entails* $\equiv$ *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)
$\langle proof \rangle$

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* $\cdot$ γ) $\Longrightarrow$
(*to-ground* (*expr* $\cdot$ (γ(*var* := *update*))), *expr′*) $\in I \Longrightarrow$
(*to-ground* (*expr* $\cdot$ 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* $\cdot$ γ) **and**
var-update: (*to-ground* (γ *var*), *to-ground update*) $\in I$
shows (*to-ground* (*expr* $\cdot$ (γ(*var* := *update*))), *expr′*) $\in I \longleftrightarrow$ (*to-ground* (*expr* $\cdot$ 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* $\cdot$ γ) *is-ground* (*expr′* $\cdot$ γ)
shows
(*to-ground* (*expr* $\cdot$ (γ(*var* := *update*))), *to-ground* (*expr′* $\cdot$ (γ(*var* := *update*)))) $\in I \longleftrightarrow$
(*to-ground* (*expr* $\cdot$ γ), *to-ground* (*expr′* $\cdot$ 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* $\Rightarrow$ *bool* **and**
empty :: *bool* **and**
connective :: *bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool* **and**
entails-def
assumes
is-negated-subst: $\bigwedge$*expr* σ. *is-negated* (*subst expr* σ) $\longleftrightarrow$ *is-negated expr* **and**
entails-def: $\bigwedge$*expr*. *entails-def expr* $\longleftrightarrow$
(*if is-negated expr then Not else* (λx. *x*))

(*Finite-Set.fold connective empty* (*sub-entails* ` *to-set expr*))
begin

notation *sub-entails* (($\models_s$ -) [*50*] *50*)
notation *entails-def* (($\models$ -) [*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* = ($\wedge$) **and** *empty* = *True*

locale *entailment-lifting-disj* = *entailment-lifting*
  where *connective* = ($\vee$) **and** *empty* = *False*

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

lemma *comp-fun-idem-conj*: *comp-fun-idem-on X* ($\wedge$)
  ⟨*proof*⟩

lemma *comp-fun-idem-disj*: *comp-fun-idem-on X* ($\vee$)
  ⟨*proof*⟩

lemma *fold-conj-insert* [*simp*]:
  Finite-Set.fold ($\wedge$) *True* (*insert b B*) $\longleftrightarrow$ *b* $\wedge$ *Finite-Set.fold* ($\wedge$) *True B*
  ⟨*proof*⟩

lemma *fold-disj-insert* [*simp*]:
  Finite-Set.fold ($\vee$) *False* (*insert b B*) $\longleftrightarrow$ *b* $\vee$ *Finite-Set.fold* ($\vee$) *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 $x \in vars\ t$
  obtains c **where** $t = c\langle Var\ x\rangle$
  ⟨*proof*⟩

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

end

context *nonground-clause*
begin

lemma *not-literal-entails* [*simp*]:
  $\neg\ upair\ `\ I \models l\ Neg\ a \longleftrightarrow upair\ `\ I \models l\ Pos\ a$
  $\neg\ upair\ `\ I \models l\ Pos\ a \longleftrightarrow upair\ `\ I \models l\ Neg\ a$
  ⟨*proof*⟩

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

end

locale *clause-entailment* = *nonground-clause* +
  fixes $I :: ('f\ gterm \times 'f\ gterm)\ set$
  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 $Upair\ t\ t' \in upair\ `\ I$
  shows $Upair\ c\langle t\rangle_G\ t'' \in\ upair\ `\ I \longleftrightarrow Upair\ c\langle t'\rangle_G\ t'' \in\ upair\ `\ I$
  ⟨*proof*⟩

lemma *upair-compatible-with-gctxtI* [*intro*]:
$Upair\ t\ t' \in upair\ `\ I \Longrightarrow Upair\ c\langle t\rangle_G\ c\langle t'\rangle_G \in upair\ `\ I$
⟨*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* = (⊙) **and** *id-subst* = *Var*
and *base-subst* = (·*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* = (·*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* = λ*C*. *upair* ‘ *I* ⊫ *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*$\langle t\rangle_G$!≈ *t''* ⟷ *upair ‘ I* $\models$*l c*$\langle t'\rangle_G$!≈ *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** ·ι *67*)

lemma *vars-inference* [*simp*]:
inference.vars (*Infer Ps C*) = ⋃ (*clause.vars ‘ set Ps*) ∪ *clause.vars C*
⟨*proof*⟩

lemma *subst-inference* [*simp*]:
Infer Ps C ·ι σ = *Infer* (*map* (λ*P*. *P* · σ) *Ps*) (*C* · σ)
⟨*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 $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $lift :: 'b \Rightarrow 'a$
  assumes *inj-lift* [*intro*]: *inj lift*
begin

definition $R_r :: 'b \Rightarrow 'b \Rightarrow bool$ **where**
  $R_r\ b\ b' \equiv R\ (lift\ b)\ (lift\ b')$

end

5.1 Strict Orders

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

abbreviation *less-eq* **where** *less-eq* $\equiv (\prec)^{==}$

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

sublocale *order* $(\preceq)$ $(\prec)$
  ⟨*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-eq*$_r$ ≡ *restriction.less-eq*
notation *less-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* (≺)

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* = (≺)
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* (≺)
begin

lemma *restricted-not-le*:
  assumes $a \in$ *restriction* $b \in$ *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* (⪯) (≺)
  ⟨*proof*⟩

end

locale *total-strict-order-restriction* =
  strict-order-restriction +

restricted-total-strict-order **where** *restriction* = *range lift* **and** *less* = (≺)
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:
⋀*t*. *t* ∈ *restriction* ⟷ *restricted t*
⋀*c*. *c* ∈ *context-restriction* ⟷ *restricted-context c*

locale *restricted-context-compatibility* =
restriction-restricted +
fixes *R Fun*
assumes
context-compatible [*simp*]:
⋀*c* t_1 t_2.
restricted t_1 ⟹
restricted t_2 ⟹
restricted-context c ⟹
R (*Fun*⟨*c*;t_1⟩) (*Fun*⟨*c*;t_2⟩) ⟷ *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*⟨*c*;t_1⟩) (*Fun*⟨*c*;t_2⟩) ⟷ *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* (≺)
  ⟨*proof*⟩

sublocale *less-eq*: *context-compatibility* (⪯)
  ⟨*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 $less\text{-}eq_t \equiv (\prec_t)^{==}$

notation $less\text{-}eq_t$ (**infix** $\preceq_t$ *50*)

end

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

lemma *reflclp-iff*: $\bigwedge R\ x\ y.\ R^{==}\ x\ y \longleftrightarrow R\ x\ y \lor x = y$

⟨*proof*⟩

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

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

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

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

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

end
theory *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 :: $'f\ \mathit{gterm} \Rightarrow {'f}\ \mathit{gterm} \Rightarrow \mathit{bool}$ +
assumes
subterm-property [*simp*]: $\bigwedge t\ c.\ c \neq \Box \Longrightarrow \mathit{less}_t\ t\ c\langle t\rangle_G$
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-eq*$_G$ ≡ *less-eq*$_r$
notation *less-eq*$_G$ (**infix** $\preceq_G$ *50*)

lemma *to-ground-less*$_r$ [*simp*]:
  assumes *is-ground e* **and** *is-ground e'*
  shows *to-ground e* $\prec_G$ *to-ground e'* ⟷ *e* ≺ *e'*
  ⟨*proof*⟩

lemma *to-ground-less-eq*$_r$ [*simp*]:

assumes *is-ground e* **and** *is-ground e′*
shows *to-ground e* $\preceq_G$ *to-ground e′* ⟷ *e* ⪯ *e′*
⟨*proof*⟩

lemma *less-eq*$_r$*-from-ground* [*simp*]:
$e_G \preceq_G e_G{}'$ ⟷ *from-ground* e_G ⪯ *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* ⟷ *expr* ≺ *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* = $base\text{-}less^{==}$ **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* ⇒ *'a multiset*
begin

definition *multiset-extension* :: *'b* ⇒ *'b* ⇒ *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* = (≺) :: *'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*. $sub \cdot_s \sigma' \preceq sub \cdot_s \sigma$

shows $expr \cdot \sigma' \preceq_m expr \cdot \sigma$
⟨*proof*⟩

lemma *less-if-all-lesseq-ex-less*:
assumes
$\forall sub \in\# to\text{-}mset\ expr.\ sub \cdot_s \sigma' \preceq sub \cdot_s \sigma$
$\exists sub \in\# to\text{-}mset\ expr.\ sub \cdot_s \sigma' \prec sub \cdot_s \sigma$
shows
$expr \cdot \sigma' \prec_m expr \cdot \sigma$
⟨*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 :: $'expr_G \Rightarrow 'sub_G\ set$
begin

sublocale *strict-order-restriction* ($\prec_m$) *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* ⇒ *'t* ⇒ *bool* +
fixes *literal-to-mset* :: *'a literal* ⇒ *'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* = λ*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* = λ*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*:
    $\bigwedge t_G\ c_G.$
      term.is-ground $t_G \Longrightarrow$
      context.is-ground $c_G \Longrightarrow$
      $c_G \neq \Box \Longrightarrow$
      $R\ t_G\ c_G\langle t_G\rangle$

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* = ($\odot$) **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* $\Rightarrow$ ($'f$, $'v$) *Term.term* $\Rightarrow$ *bool*
begin

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

sublocale *base-grounded-order* **where**
comp-subst = ($\odot$) **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* = (⊙) **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 $term.order.less\text{-}eq_G$ (**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*⟩

lemma $less_{lG}$*-rewrite* [*simp*]: *multiset-extension.multiset-extension* $(\prec_{tG})$ *mset-lit* $= (\prec_{lG})$
⟨*proof*⟩

lemma $less_{cG}$*-rewrite* [*simp*]:
multiset-extension.multiset-extension $(\prec_{lG})$ $(\lambda x.\ x) = (\prec_{cG})$
⟨*proof*⟩

lemma *is-maximal-rewrite* [*simp*]:
is-maximal-in-mset-wrt $(\prec_{lG})$ C l = *is-maximal* (*literal.from-ground* l) (*clause.from-ground* C)
⟨*proof*⟩

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

lemma *is-strictly-maximal-rewrite* [*simp*]:
is-strictly-maximal-in-mset-wrt $(\prec_{lG})$ C l =
is-strictly-maximal (*literal.from-ground* l) (*clause.from-ground* C)
⟨*proof*⟩

sublocale *ground*: *ground-order-with-equality* **where**

$less_t = (\prec_{tG})$
rewrites
multiset-extension.multiset-extension $(\prec_{tG})$ *mset-lit* $= (\prec_{lG})$ **and**
multiset-extension.multiset-extension $(\prec_{lG})$ $(\lambda x.\ x) = (\prec_{cG})$ **and**
$\bigwedge l\ C.$ *ground.is-maximal* $l\ C \longleftrightarrow$ *is-maximal* (*literal.from-ground* l) (*clause.from-ground* C) **and**
$\bigwedge l\ C.$ *ground.is-strictly-maximal* $l\ C \longleftrightarrow$ *is-strictly-maximal* (*literal.from-ground* l) (*clause.from-ground* C)
⟨*proof*⟩

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)

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
Functional-Substitution-Typing
Typed-Functional-Substitution
Abstract-Substitution.Functional-Substitution-Example

begin

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

Inductive predicates defining well-typed terms.

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

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

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

global-interpretation *term*: *base-functional-substitution-typing* **where**
  typed = *typed* ($\mathcal{F}$ :: ('*f*, '*ty*) *fun-types*) **and** *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* ⇒ '*v set*
  ⟨*proof*⟩

A selection of substitution properties for typed terms.

locale *typed-term-subst-properties* =
  typed: *explicitly-typed-subst-stability* **where** *typed* = *typed* $\mathcal{F}$ +
  welltyped: *explicitly-typed-subst-stability* **where** *typed* = *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* ⇒ '*v set* **and** $\mathcal{F}$ = $\mathcal{F}$
for $\mathcal{F}$ :: '*f* ⇒ '*ty list* × '*ty*
⟨*proof*⟩

Examples of generated lemmas and definitions

thm
  term.welltyped.right-unique
  term.welltyped.explicit-subst-stability
  term.welltyped.subst-stability
  term.welltyped.subst-update

  term.typed.right-unique

term.typed.explicit-subst-stability
term.typed.subst-stability
term.typed.subst-update

term.is-welltyped-on-subset
term.is-typed-on-subset
term.is-welltyped-id-subst
term.is-typed-id-subst

term *term.is-welltyped*
term *term.subst.is-welltyped-on*
term *term.subst.is-welltyped*
term *term.is-typed*
term *term.subst.is-typed-on*
term *term.subst.is-typed*

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

All property locales have corresponding lifting locales

locale *nonground-uniform-typing-lifting* =
functional-substitution-uniform-typing-lifting **where**
base-typed = *typed* $\mathcal{F}$ **and** *base-welltyped* = *welltyped* $\mathcal{F}$ +

is-typed: *uniform-typed-subst-stability-lifting* **where**
base-typed = *typed* $\mathcal{F}$ +

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

locale *nonground-typing-lifting* =
functional-substitution-typing-lifting **where**
base-typed = *typed* $\mathcal{F}$ **and** *base-welltyped* = *welltyped* $\mathcal{F}$ +

is-typed: *typed-subst-stability-lifting* **where** *base-typed* = *typed* $\mathcal{F}$ +

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

locale *example-typing-lifting* =

fixes $\mathcal{F}$ :: ('f, 'ty) *fun-types*
begin

sublocale *equation*:
uniform-typing-lifting **where**
sub-typed = *typed* $\mathcal{F}$ $\mathcal{V}$ **and** *sub-welltyped* = *welltyped* $\mathcal{F}$ $\mathcal{V}$ **and**
to-set = *set-prod*
⟨*proof*⟩

sublocale *equation*:
nonground-uniform-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*
⟨*proof*⟩

Lifted lemmas and definitions

thm
equation.is-welltyped-def
equation.is-typed-def

equation.is-welltyped.subst-stability
equation.is-typed.subst-stability
equation.is-typed-if-is-welltyped

We can lift multiple levels

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

Lifted lemmas and definitions

thm
equation-set.is-welltyped-def
equation-set.is-typed-def

equation-set.is-welltyped.subst-stability
equation-set.is-typed.subst-stability

equation-set.is-typed-if-is-welltyped

end

Interpretation with Unit-Typing

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

end