Abstract Substituions as Monoid Actions

Martin Desharnais

October 4, 2024

Abstract

This entry provides a small, reusable, theory that specifies the abstract concept of substition as monoid action. Both the substitution type and the object type are kept abstract. The theory provides multiple useful definitions and lemmas. Two example usages are provided for first order terms: one for terms from the AFP/First_Order_Terms session and one for terms from the Isabelle/HOL-ex session.

Contents

theory *Substitution*
  imports *Main*

begin

1 General Results on Groups

lemma (**in** *monoid*) *right-inverse-idem*:
  fixes *inv*
  assumes *right-inverse*: $\bigwedge a.\ a * inv\ a = \mathbf{1}$
  shows $\bigwedge a.\ inv\ (inv\ a) = a$
    ⟨*proof*⟩

lemma (**in** *monoid*) *left-inverse-if-right-inverse*:
  fixes *inv*
  assumes
    right-inverse: $\bigwedge a.\ a * inv\ a = \mathbf{1}$
  shows $inv\ a * a = \mathbf{1}$
  ⟨*proof*⟩

lemma (**in** *monoid*) *group-wrt-right-inverse*:
  fixes *inv*
  assumes *right-inverse*: $\bigwedge a.\ a * inv\ a = \mathbf{1}$
  shows *group* $(*)$ $\mathbf{1}$ *inv*
⟨*proof*⟩

2 Semigroup Action

We define both left and right semigroup actions. Left semigroup actions seem to be prevalent in algebra, but right semigroup actions directly uses the usual notation of term/atom/literal/clause substitution.

locale *left-semigroup-action* = *semigroup* +
  fixes *action* :: $'a \Rightarrow 'b \Rightarrow 'b$ (**infix** $\cdot$ *70*)
  assumes *action-compatibility*[*simp*]: $\bigwedge a\ b\ x.\ (a * b) \cdot x = a \cdot (b \cdot x)$

locale *right-semigroup-action* = *semigroup* +
  fixes *action* :: $'b \Rightarrow 'a \Rightarrow 'b$ (**infix** $\cdot$ *70*)
  assumes *action-compatibility*[*simp*]: $\bigwedge x\ a\ b.\ x \cdot (a * b) = (x \cdot a) \cdot b$

We then instantiate the right action in the context of the left action in order to get access to any lemma proven in the context of the other locale. We do analogously in the context of the right locale.

sublocale *left-semigroup-action* ⊆ *right*: *right-semigroup-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
⟨*proof*⟩

sublocale *right-semigroup-action* ⊆ *left*: *left-semigroup-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
⟨*proof*⟩

lemma (**in** *right-semigroup-action*) *lifting-semigroup-action-to-set*:
  right-semigroup-action $(*)$ $(\lambda X\ a.\ (\lambda x.\ action\ x\ a)$ ‘ $X)$
⟨*proof*⟩

lemma (**in** *right-semigroup-action*) *lifting-semigroup-action-to-list*:
  right-semigroup-action $(*)$ $(\lambda xs\ a.\ map\ (\lambda x.\ action\ x\ a)\ xs)$
⟨*proof*⟩

3 Monoid Action

locale *left-monoid-action* = *monoid* +
  fixes *action* :: $'a \Rightarrow 'b \Rightarrow 'b$ (**infix** $\cdot$ *70*)
  assumes
    monoid-action-compatibility: $\bigwedge a\ b\ x.\ (a * b) \cdot x = a \cdot (b \cdot x)$ **and**
    action-neutral[*simp*]: $\bigwedge x.\ \mathbf{1} \cdot x = x$

locale *right-monoid-action* = *monoid* +
  fixes *action* :: $'b \Rightarrow 'a \Rightarrow 'b$ (**infix** $\cdot$ *70*)
  assumes
    monoid-action-compatibility: $\bigwedge x\ a\ b.\ x \cdot (a * b) = (x \cdot a) \cdot b$ **and**
    action-neutral[*simp*]: $\bigwedge x.\ x \cdot \mathbf{1} = x$

sublocale *left-monoid-action* ⊆ *left-semigroup-action*
  ⟨*proof*⟩

sublocale *right-monoid-action* ⊆ *right-semigroup-action*
  ⟨*proof*⟩

sublocale *left-monoid-action* ⊆ *right*: *right-monoid-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
  ⟨*proof*⟩

sublocale *right-monoid-action* ⊆ *left*: *left-monoid-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
  ⟨*proof*⟩

lemma (**in** *right-monoid-action*) *lifting-monoid-action-to-set*:
  right-monoid-action $(*)$ $\mathbf{1}$ $(\lambda X\ a.\ (\lambda x.\ action\ x\ a)$ ‘ $X)$
⟨*proof*⟩

lemma (**in** *right-monoid-action*) *lifting-monoid-action-to-list*:
  right-monoid-action $(*)$ $\mathbf{1}$ $(\lambda xs\ a.\ map\ (\lambda x.\ action\ x\ a)\ xs)$
⟨*proof*⟩

4 Group Action

locale *left-group-action* = *group* +
  fixes *action* :: $'a \Rightarrow 'b \Rightarrow 'b$ (**infix** $\cdot$ *70*)

assumes
group-action-compatibility: $\bigwedge a\ b\ x.\ (a * b) \cdot x = a \cdot (b \cdot x)$ **and**
group-action-neutral: $\bigwedge x.\ \mathbf{1} \cdot x = x$

locale *right-group-action* = *group* +
fixes *action* :: $'b \Rightarrow 'a \Rightarrow 'b$ (**infixl** $\cdot$ *70*)
assumes
group-action-compatibility: $\bigwedge x\ a\ b.\ x \cdot (a * b) = (x \cdot a) \cdot b$ **and**
group-action-neutral: $\bigwedge x.\ x \cdot \mathbf{1} = x$

sublocale *left-group-action* ⊆ *left-monoid-action*
⟨*proof*⟩

sublocale *right-group-action* ⊆ *right-monoid-action*
⟨*proof*⟩

sublocale *left-group-action* ⊆ *right*: *right-group-action* **where**
$f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
⟨*proof*⟩

sublocale *right-group-action* ⊆ *left*: *left-group-action* **where**
$f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
⟨*proof*⟩

5 Assumption-free Substitution

locale *substitution-ops* =
fixes
subst :: $'x \Rightarrow 's \Rightarrow 'x$ (**infixl** $\cdot$ *67*) **and**
id-subst :: $'s$ **and**
comp-subst :: $'s \Rightarrow 's \Rightarrow 's$ (**infixl** $\odot$ *67*) **and**
is-ground :: $'x \Rightarrow bool$
begin

definition *subst-set* :: $'x\ set \Rightarrow 's \Rightarrow 'x\ set$ **where**
subst-set $X\ \sigma = (\lambda x.\ subst\ x\ \sigma)$ ‘ X

definition *subst-list* :: $'x\ list \Rightarrow 's \Rightarrow 'x\ list$ **where**
subst-list $xs\ \sigma = map\ (\lambda x.\ subst\ x\ \sigma)\ xs$

definition *is-ground-set* :: $'x\ set \Rightarrow bool$ **where**
is-ground-set $X \longleftrightarrow (\forall x \in X.\ is\text{-}ground\ x)$

definition *is-ground-subst* :: $'s \Rightarrow bool$ **where**
is-ground-subst $\gamma \longleftrightarrow (\forall x.\ is\text{-}ground\ (x \cdot \gamma))$

definition *generalizes* :: $'x \Rightarrow 'x \Rightarrow bool$ **where**
generalizes $x\ y \longleftrightarrow (\exists \sigma.\ x \cdot \sigma = y)$

definition *specializes* :: $'x \Rightarrow 'x \Rightarrow$ *bool* **where**
specializes $x\ y \equiv$ *generalizes* $y\ x$

definition *strictly-generalizes* :: $'x \Rightarrow 'x \Rightarrow$ *bool* **where**
strictly-generalizes $x\ y \longleftrightarrow$ *generalizes* $x\ y \wedge \neg$ *generalizes* $y\ x$

definition *strictly-specializes* :: $'x \Rightarrow 'x \Rightarrow$ *bool* **where**
strictly-specializes $x\ y \equiv$ *strictly-generalizes* $y\ x$

definition *instances* :: $'x \Rightarrow 'x$ *set* **where**
instances $x = \{y.$ *generalizes* $x\ y\}$

definition *instances-set* :: $'x$ *set* $\Rightarrow 'x$ *set* **where**
instances-set $X = (\bigcup x \in X.$ *instances* $x)$

definition *ground-instances* :: $'x \Rightarrow 'x$ *set* **where**
ground-instances $x = \{x_{\mathcal{G}} \in$ *instances* $x.$ *is-ground* $x_{\mathcal{G}}\}$

definition *ground-instances-set* :: $'x$ *set* $\Rightarrow 'x$ *set* **where**
ground-instances-set $X = \{x_{\mathcal{G}} \in$ *instances-set* $X.$ *is-ground* $x_{\mathcal{G}}\}$

lemma *ground-instances-set-eq-Union-ground-instances*:
ground-instances-set $X = (\bigcup x \in X.$ *ground-instances* $x)$
⟨*proof*⟩

lemma *ground-instances-eq-Collect-subst-grounding*:
ground-instances $x = \{x \cdot \gamma \mid \gamma.$ *is-ground* $(x \cdot \gamma)\}$
⟨*proof*⟩

definition *is-renaming* :: $'s \Rightarrow$ *bool* **where**
is-renaming $\varrho \longleftrightarrow (\exists\, \varrho$-*inv*. $\varrho \odot \varrho$-*inv* $=$ *id-subst*$)$

definition *renaming-inverse* **where**
is-renaming $\varrho \Longrightarrow$ *renaming-inverse* $\varrho = ($*SOME* ϱ-*inv*. $\varrho \odot \varrho$-*inv* $=$ *id-subst*$)$

lemma *renaming-comp-renaming-inverse*[*simp*]:
is-renaming $\varrho \Longrightarrow \varrho \odot$ *renaming-inverse* $\varrho =$ *id-subst*
⟨*proof*⟩

definition *is-unifier* :: $'s \Rightarrow 'x$ *set* $\Rightarrow$ *bool* **where**
is-unifier $\upsilon\ X \longleftrightarrow$ *card* $($*subst-set* $X\ \upsilon) \leq 1$

definition *is-unifier-set* :: $'s \Rightarrow 'x$ *set set* $\Rightarrow$ *bool* **where**
is-unifier-set $\upsilon\ XX \longleftrightarrow (\forall X \in XX.$ *is-unifier* $\upsilon\ X)$

definition *is-mgu* :: $'s \Rightarrow 'x$ *set set* $\Rightarrow$ *bool* **where**
is-mgu $\mu\ XX \longleftrightarrow$ *is-unifier-set* $\mu\ XX \wedge (\forall \upsilon.$ *is-unifier-set* $\upsilon\ XX \longrightarrow (\exists \sigma.\ \mu \odot \sigma = \upsilon))$

definition *is-imgu* :: *'s* ⇒ *'x set set* ⇒ *bool* **where**
is-imgu μ *XX* ⟷ *is-unifier-set* μ *XX* ∧ (∀ τ. *is-unifier-set* τ *XX* ⟶ $\mu \odot \tau = \tau$)

definition *is-idem* :: *'s* ⇒ *bool* **where**
is-idem σ ⟷ $\sigma \odot \sigma = \sigma$

lemma *is-unifier-iff-if-finite*:
assumes *finite X*
shows *is-unifier* σ *X* ⟷ (∀ *x*∈*X*. ∀ *y*∈*X*. $x \cdot \sigma = y \cdot \sigma$)
⟨*proof*⟩

lemma *is-unifier-singleton*[*simp*]: *is-unifier* υ {*x*}
⟨*proof*⟩

lemma *is-unifier-set-insert-singleton*[*simp*]:
is-unifier-set σ (*insert* {*x*} *XX*) ⟷ *is-unifier-set* σ *XX*
⟨*proof*⟩

lemma *is-mgu-insert-singleton*[*simp*]: *is-mgu* μ (*insert* {*x*} *XX*) ⟷ *is-mgu* μ *XX*
⟨*proof*⟩

lemma *is-imgu-insert-singleton*[*simp*]: *is-imgu* μ (*insert* {*x*} *XX*) ⟷ *is-imgu* μ *XX*
⟨*proof*⟩

lemma *subst-set-empty*[*simp*]: *subst-set* {} σ = {}
⟨*proof*⟩

lemma *subst-set-insert*[*simp*]: *subst-set* (*insert x X*) σ = *insert* ($x \cdot \sigma$) (*subst-set X* σ)
⟨*proof*⟩

lemma *subst-set-union*[*simp*]: *subst-set* (*X1* ∪ *X2*) σ = *subst-set X1* σ ∪ *subst-set X2* σ
⟨*proof*⟩

lemma *subst-list-Nil*[*simp*]: *subst-list* [] σ = []
⟨*proof*⟩

lemma *subst-list-insert*[*simp*]: *subst-list* (*x* # *xs*) σ = ($x \cdot \sigma$) # (*subst-list xs* σ)
⟨*proof*⟩

lemma *subst-list-append*[*simp*]: *subst-list* (xs_1 @ xs_2) σ = *subst-list* xs_1 σ @ *subst-list* xs_2 σ
⟨*proof*⟩

lemma *is-unifier-set-union*:

is-unifier-set υ $(XX_1 \cup XX_2) \longleftrightarrow$ *is-unifier-set* υ $XX_1 \wedge$ *is-unifier-set* υ XX_2
⟨*proof*⟩

lemma *is-unifier-subset*: *is-unifier* υ $A \Longrightarrow$ *finite* $A \Longrightarrow B \subseteq A \Longrightarrow$ *is-unifier* υ B
⟨*proof*⟩

lemma *is-ground-set-subset*: *is-ground-set* $A \Longrightarrow B \subseteq A \Longrightarrow$ *is-ground-set* B
⟨*proof*⟩

lemma *is-ground-set-ground-instances*[*simp*]: *is-ground-set* (*ground-instances* x)
⟨*proof*⟩

lemma *is-ground-set-ground-instances-set*[*simp*]: *is-ground-set* (*ground-instances-set* x)
⟨*proof*⟩

end

6 Basic Substitution

locale *substitution* =
comp-subst: *right-monoid-action comp-subst id-subst subst* +
substitution-ops subst id-subst comp-subst is-ground
for
comp-subst :: $'s \Rightarrow 's \Rightarrow 's$ (**infixl** $\odot$ *70*) **and**
id-subst :: $'s$ **and**
subst :: $'x \Rightarrow 's \Rightarrow 'x$ (**infixl** $\cdot$ *70*) **and**

— Predicate identifying the fixed elements w.r.t. the monoid action
is-ground :: $'x \Rightarrow$ *bool* +
assumes
all-subst-ident-if-ground: *is-ground* $x \Longrightarrow (\forall \sigma.\ x \cdot \sigma = x)$
begin

sublocale *comp-subst-set*: *right-monoid-action comp-subst id-subst subst-set*
⟨*proof*⟩

sublocale *comp-subst-list*: *right-monoid-action comp-subst id-subst subst-list*
⟨*proof*⟩

6.1 Substitution Composition

lemmas *subst-comp-subst* = *comp-subst.action-compatibility*
lemmas *subst-set-comp-subst* = *comp-subst-set.action-compatibility*
lemmas *subst-list-comp-subst* = *comp-subst-list.action-compatibility*

6.2 Substitution Identity

lemmas *subst-id-subst* = *comp-subst.action-neutral*
lemmas *subst-set-id-subst* = *comp-subst-set.action-neutral*
lemmas *subst-list-id-subst* = *comp-subst-list.action-neutral*

lemma *is-renaming-id-subst*[*simp*]: *is-renaming id-subst*
⟨*proof*⟩

lemma *is-unifier-id-subst-empty*[*simp*]: *is-unifier id-subst* {}
⟨*proof*⟩

lemma *is-unifier-set-id-subst-empty*[*simp*]: *is-unifier-set id-subst* {}
⟨*proof*⟩

lemma *is-mgu-id-subst-empty*[*simp*]: *is-mgu id-subst* {}
⟨*proof*⟩

lemma *is-imgu-id-subst-empty*[*simp*]: *is-imgu id-subst* {}
⟨*proof*⟩

lemma *is-idem-id-subst*[*simp*]: *is-idem id-subst*
⟨*proof*⟩

lemma *is-unifier-id-subst*: *is-unifier id-subst X* ⟷ *card X* ≤ *1*
⟨*proof*⟩

lemma *is-unifier-set-id-subst*: *is-unifier-set id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
⟨*proof*⟩

lemma *is-mgu-id-subst*: *is-mgu id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
⟨*proof*⟩

lemma *is-imgu-id-subst*: *is-imgu id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
⟨*proof*⟩

6.3 Generalization

sublocale *generalizes*: *preorder generalizes strictly-generalizes*
⟨*proof*⟩

6.4 Substituting on Ground Expressions

lemma *subst-ident-if-ground*[*simp*]: *is-ground x* ⟹ *x* · σ = *x*
⟨*proof*⟩

lemma *subst-set-ident-if-ground*[*simp*]: *is-ground-set X* ⟹ *subst-set X* σ = *X*
⟨*proof*⟩

6.5 Instances of Ground Expressions

lemma *instances-ident-if-ground*[*simp*]: *is-ground* $x \Longrightarrow$ *instances* $x = \{x\}$
⟨*proof*⟩

lemma *instances-set-ident-if-ground*[*simp*]: *is-ground-set* $X \Longrightarrow$ *instances-set* $X = X$
⟨*proof*⟩

lemma *ground-instances-ident-if-ground*[*simp*]: *is-ground* $x \Longrightarrow$ *ground-instances* $x = \{x\}$
⟨*proof*⟩

lemma *ground-instances-set-ident-if-ground*[*simp*]: *is-ground-set* $X \Longrightarrow$ *ground-instances-set* $X = X$
⟨*proof*⟩

6.6 Unifier of Ground Expressions

lemma *ground-eq-ground-if-unifiable*:
assumes *is-unifier* υ $\{t_1, t_2\}$ **and** *is-ground* t_1 **and** *is-ground* t_2
shows $t_1 = t_2$
⟨*proof*⟩

lemma *ball-eq-constant-if-unifier*:
assumes *finite* X **and** $x \in X$ **and** *is-unifier* υ X **and** *is-ground-set* X
shows $\forall\, y \in X.\ y = x$
⟨*proof*⟩

lemma *subst-mgu-eq-subst-mgu*:
assumes *is-mgu* μ $\{\{t_1, t_2\}\}$
shows $t_1 \cdot \mu = t_2 \cdot \mu$
⟨*proof*⟩

lemma *subst-imgu-eq-subst-imgu*:
assumes *is-imgu* μ $\{\{t_1, t_2\}\}$
shows $t_1 \cdot \mu = t_2 \cdot \mu$
⟨*proof*⟩

6.7 Ground Substitutions

lemma *is-ground-subst-comp-left*: *is-ground-subst* $\sigma \Longrightarrow$ *is-ground-subst* $(\sigma \odot \tau)$
⟨*proof*⟩

lemma *is-ground-subst-comp-right*: *is-ground-subst* $\tau \Longrightarrow$ *is-ground-subst* $(\sigma \odot \tau)$
⟨*proof*⟩

lemma *is-ground-subst-is-ground*:
assumes *is-ground-subst* γ
shows *is-ground* $(t \cdot \gamma)$

⟨*proof*⟩

6.8 IMGU is Idempotent and an MGU

lemma *is-imgu-iff-is-idem-and-is-mgu*: *is-imgu* μ *XX* $\longleftrightarrow$ *is-idem* $\mu \wedge$ *is-mgu* μ *XX*
⟨*proof*⟩

6.9 IMGU can be used before unification

lemma *subst-imgu-subst-unifier*:
assumes *unif*: *is-unifier* υ X **and** *imgu*: *is-imgu* μ $\{X\}$ **and** $x \in X$
shows $x \cdot \mu \cdot \upsilon = x \cdot \upsilon$
⟨*proof*⟩

6.10 Groundings Idempotence

lemma *image-ground-instances-ground-instances*:
ground-instances ‘ *ground-instances* $x = (\lambda x.\ \{x\})$ ‘ *ground-instances* x
⟨*proof*⟩

lemma *grounding-of-set-grounding-of-set-idem*[*simp*]:
ground-instances-set (*ground-instances-set* X) = *ground-instances-set* X
⟨*proof*⟩

6.11 Instances of Substitution

lemma *instances-subst*:
instances $(x \cdot \sigma) \subseteq$ *instances* x
⟨*proof*⟩

lemma *instances-set-subst-set*:
instances-set (*subst-set* X σ) $\subseteq$ *instances-set* X
⟨*proof*⟩

lemma *ground-instances-subst*:
ground-instances $(x \cdot \sigma) \subseteq$ *ground-instances* x
⟨*proof*⟩

lemma *ground-instances-set-subst-set*:
ground-instances-set (*subst-set* X σ) $\subseteq$ *ground-instances-set* X
⟨*proof*⟩

6.12 Instances of Renamed Expressions

lemma *instances-subst-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *instances* $(x \cdot \varrho)$ = *instances* x
⟨*proof*⟩

lemma *instances-set-subst-set-ident-if-renaming*[*simp*]:

is-renaming $\varrho \Longrightarrow$ *instances-set* (*subst-set* X ϱ) = *instances-set* X
⟨*proof*⟩

lemma *ground-instances-subst-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *ground-instances* ($x \cdot \varrho$) = *ground-instances* x
⟨*proof*⟩

lemma *ground-instances-set-subst-set-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *ground-instances-set* (*subst-set* X ϱ) = *ground-instances-set* X
⟨*proof*⟩

end

end
theory *Substitution-First-Order-Term*
imports
Substitution
First-Order-Terms.Unification
begin

abbreviation *is-ground-trm* **where**
is-ground-trm $t \equiv$ *vars-term* $t = \{\}$

lemma *is-ground-iff*: *is-ground-trm* ($t \cdot \gamma$) $\longleftrightarrow$ ($\forall x \in$ *vars-term* t. *is-ground-trm* (γ x))
⟨*proof*⟩

lemma *is-ground-trm-iff-ident-forall-subst*: *is-ground-trm* $t \longleftrightarrow (\forall \sigma.\ t \cdot \sigma = t)$
⟨*proof*⟩

global-interpretation *term-subst*: *substitution* **where**
subst = *subst-apply-term* **and** *id-subst* = *Var* **and** *comp-subst* = *subst-compose*
and
is-ground = *is-ground-trm*
⟨*proof*⟩

lemma *term-subst-is-unifier-iff-unifiers*:
assumes *finite* X
shows *term-subst.is-unifier* μ $X \longleftrightarrow \mu \in$ *unifiers* ($X \times X$)
⟨*proof*⟩

lemma *term-subst-is-unifier-set-iff-unifiers*:
assumes $\forall X \in XX$. *finite* X
shows *term-subst.is-unifier-set* μ $XX \longleftrightarrow \mu \in$ *unifiers* ($\bigcup X \in XX.\ X \times X$)
⟨*proof*⟩

lemma *term-subst-is-imgu-iff-is-imgu*:
assumes $\forall X \in XX$. *finite* X
shows *term-subst.is-imgu* μ $XX \longleftrightarrow$ *is-imgu* μ ($\bigcup X \in XX.\ X \times X$)

⟨*proof*⟩

lemma *range-vars-subset-if-is-imgu*:
assumes *term-subst.is-imgu* μ *XX* $\forall X{\in}XX.$ *finite X finite XX*
shows *range-vars* $\mu \subseteq (\bigcup t{\in}\bigcup XX.$ *vars-term t*)
⟨*proof*⟩

lemma *term-subst-is-renaming-iff*:
term-subst.is-renaming $\varrho \longleftrightarrow$ *inj* $\varrho \wedge (\forall x.$ *is-Var* $(\varrho\ x))$
⟨*proof*⟩

lemma *term-subst-is-renaming-iff-ex-inj-fun-on-vars*:
term-subst.is-renaming $\varrho \longleftrightarrow (\exists f.$ *inj f* $\wedge\ \varrho =$ *Var* $\circ f)$
⟨*proof*⟩

lemma *ground-imgu-equals*:
assumes *is-ground-trm* t_1 **and** *is-ground-trm* t_2 **and** *term-subst.is-imgu* μ $\{\{t_1, t_2\}\}$
shows $t_1 = t_2$
⟨*proof*⟩

lemma *the-mgu-is-unifier*:
assumes *term* $\cdot$ *the-mgu term term'* = *term'* $\cdot$ *the-mgu term term'*
shows *term-subst.is-unifier* (*the-mgu term term'*) {*term*, *term'*}
⟨*proof*⟩

lemma *imgu-exists-extendable*:
fixes υ :: ('*f*, '*v*) *subst*
assumes *term* $\cdot\ \upsilon$ = *term'* $\cdot\ \upsilon$ *P term term'* (*the-mgu term term'*)
obtains μ :: ('*f*, '*v*) *subst*
where $\upsilon = \mu \circ_s \upsilon$ *term-subst.is-imgu* μ {{*term*, *term'*}} *P term term'* μ
⟨*proof*⟩

lemma *imgu-exists*:
fixes υ :: ('*f*, '*v*) *subst*
assumes *term* $\cdot\ \upsilon$ = *term'* $\cdot\ \upsilon$
obtains μ :: ('*f*, '*v*) *subst*
where $\upsilon = \mu \circ_s \upsilon$ *term-subst.is-imgu* μ {{*term*, *term'*}}
⟨*proof*⟩

lemma *is-renaming-if-term-subst-is-renaming*:
assumes *term-subst.is-renaming* ϱ
shows *is-renaming* ϱ
⟨*proof*⟩

end
theory *Substitution-HOL-ex-Unification*
imports

Substitution
HOL−ex.Unification
begin

no-notation *Comb* (**infix** · *60*)

quotient-type $'a$ *subst* = ($'a$ × $'a$ *trm*) *list* / ($\doteq$)
⟨*proof*⟩

lift-definition *subst-comp* :: $'a$ *subst* ⇒ $'a$ *subst* ⇒ $'a$ *subst* (**infixl** ⊙ *67*)
is *Unification.comp*
⟨*proof*⟩

definition *subst-id* :: $'a$ *subst* **where**
subst-id = *abs-subst* []

global-interpretation *subst-comp*: *monoid subst-comp subst-id*
⟨*proof*⟩

lift-definition *subst-apply* :: $'a$ *trm* ⇒ $'a$ *subst* ⇒ $'a$ *trm*
is *Unification.subst*
⟨*proof*⟩

abbreviation *is-ground-trm* **where**
is-ground-trm t ≡ *vars-of t* = {}

global-interpretation *term-subst*: *substitution* **where**
subst = *subst-apply* **and** *id-subst* = *subst-id* **and** *comp-subst* = *subst-comp* **and**
is-ground = *is-ground-trm*
⟨*proof*⟩

end