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 $\bigwedge a.\ inv\ (inv\ a) = a$
    by (*metis assoc right-inverse right-neutral*)

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

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 *unfold-locales*
  show $\bigwedge a.\ \mathbf{1} * a = a$
    by *simp*
next
  show $\bigwedge a.\ inv\ a * a = \mathbf{1}$
    by (*metis left-inverse-if-right-inverse right-inverse*)
qed

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* $\subseteq$ *right*: *right-semigroup-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$

proof *unfold-locales*
  show $\bigwedge a\ b\ c.\ c * (b * a) = c * b * a$
    by (*simp only*: *assoc*)
next
  show $\bigwedge x\ a\ b.\ (b * a) \cdot x = b \cdot (a \cdot x)$
    by *simp*
qed

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 *unfold-locales*
  show $\bigwedge a\ b\ c.\ c * (b * a) = c * b * a$
    by (*simp only*: *assoc*)
next
  show $\bigwedge a\ b\ x.\ x \cdot (b * a) = (x \cdot b) \cdot a$
    by *simp*
qed

lemma (**in** *right-semigroup-action*) *lifting-semigroup-action-to-set*:
  right-semigroup-action $(*)\ (\lambda X\ a.\ (\lambda x.\ action\ x\ a)\ `\ X)$
proof *unfold-locales*
  show $\bigwedge x\ a\ b.\ (\lambda x.\ x \cdot (a * b))\ `\ x = (\lambda x.\ x \cdot b)\ `\ (\lambda x.\ x \cdot a)\ `\ x$
    by (*simp add*: *image-comp*)
qed

lemma (**in** *right-semigroup-action*) *lifting-semigroup-action-to-list*:
  right-semigroup-action $(*)\ (\lambda xs\ a.\ map\ (\lambda x.\ action\ x\ a)\ xs)$
proof *unfold-locales*
  show $\bigwedge x\ a\ b.\ map\ (\lambda x.\ x \cdot (a * b))\ x = map\ (\lambda x.\ x \cdot b)\ (map\ (\lambda x.\ x \cdot a)\ x)$
    by (*simp add*: *image-comp*)
qed

3 Monoid Action

locale *left-monoid-action* = *monoid* +
  fixes *action* :: $'a \Rightarrow 'b \Rightarrow 'b$ (**infix** · *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** · *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*
  by *unfold-locales* (*fact monoid-action-compatibility*)

sublocale *right-monoid-action* $\subseteq$ *right-semigroup-action*
  by *unfold-locales* (*fact monoid-action-compatibility*)

sublocale *left-monoid-action* $\subseteq$ *right*: *right-monoid-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
  by *unfold-locales simp-all*

sublocale *right-monoid-action* $\subseteq$ *left*: *left-monoid-action* **where**
  $f = \lambda x\ y.\ f\ y\ x$ **and** $action = \lambda x\ y.\ action\ y\ x$
  by *unfold-locales simp-all*

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 (*unfold-locales*)
  show $\bigwedge x\ a\ b.\ (\lambda x.\ x \cdot (a * b))$ ‘ $x = (\lambda x.\ x \cdot b)$ ‘ $(\lambda x.\ x \cdot a)$ ‘ x
    by (*simp add*: *image-comp*)
next
  show $\bigwedge x.\ (\lambda x.\ x \cdot \mathbf{1})$ ‘ $x = x$
    by *simp*
qed

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 *unfold-locales*
  show $\bigwedge x\ a\ b.\ map\ (\lambda x.\ x \cdot (a * b))\ x = map\ (\lambda x.\ x \cdot b)\ (map\ (\lambda x.\ x \cdot a)\ x)$
    by *simp*
next
  show $\bigwedge x.\ map\ (\lambda x.\ x \cdot \mathbf{1})\ x = x$
    by *simp*
qed

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* $\subseteq$ *left-monoid-action*
  by *unfold-locales* (*fact group-action-compatibility group-action-neutral*)+

sublocale *right-group-action* $\subseteq$ *right-monoid-action*

by *unfold-locales* (*fact group-action-compatibility group-action-neutral*)+

sublocale *left-group-action* $\subseteq$ *right*: *right-group-action* **where**
f = λ*x y*. *f y x* **and** *action* = λ*x y*. *action y x*
by *unfold-locales simp-all*

sublocale *right-group-action* $\subseteq$ *left*: *left-group-action* **where**
f = λ*x y*. *f y x* **and** *action* = λ*x y*. *action y x*
by *unfold-locales simp-all*

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 σ = (λ*x*. *subst x* σ) ‘ *X*

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

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

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

definition *generalizes* :: $'x \Rightarrow 'x \Rightarrow$ *bool* **where**
generalizes x y $\longleftrightarrow$ ($\exists$ σ. *x* $\cdot$ σ = *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* $x)$
unfolding *ground-instances-set-def ground-instances-def*
unfolding *instances-set-def*
by *auto*

lemma *ground-instances-eq-Collect-subst-grounding*:
ground-instances $x = \{x \cdot \gamma \mid \gamma.$ *is-ground* $(x \cdot \gamma)\}$
by (*auto simp*: *ground-instances-def instances-def generalizes-def*)

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* ϱ = (*SOME* ϱ-*inv*. $\varrho \odot \varrho$-*inv* = *id-subst*)

lemma *renaming-comp-renaming-inverse*[*simp*]:
is-renaming $\varrho \Longrightarrow \varrho \odot$ *renaming-inverse* ϱ = *id-subst*
by (*auto simp*: *is-renaming-def renaming-inverse-def intro*: *someI-ex*)

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

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

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

definition *is-imgu* :: $'s \Rightarrow 'x$ *set set* $\Rightarrow$ *bool* **where**
is-imgu μ $XX \longleftrightarrow$ *is-unifier-set* μ $XX \land (\forall \tau.$ *is-unifier-set* τ $XX \longrightarrow \mu \odot \tau = \tau)$

definition *is-idem* :: $'s \Rightarrow$ *bool* **where**
is-idem $\sigma \longleftrightarrow \sigma \odot \sigma = \sigma$

lemma *is-unifier-iff-if-finite*:
assumes *finite* X
shows *is-unifier* σ $X \longleftrightarrow (\forall x{\in}X.\ \forall y{\in}X.\ x \cdot \sigma = y \cdot \sigma)$
proof (*rule iffI*)

show *is-unifier* σ $X \Longrightarrow (\forall x{\in}X.\ \forall y{\in}X.\ x \cdot \sigma = y \cdot \sigma)$
using *assms*
unfolding *is-unifier-def*
by (*metis One-nat-def card-le-Suc0-iff-eq finite-imageI image-eqI subst-set-def*)
next
show $(\forall x{\in}X.\ \forall y{\in}X.\ x \cdot \sigma = y \cdot \sigma) \Longrightarrow$ *is-unifier* σ X
unfolding *is-unifier-def*
by (*smt* (*verit*, *del-insts*) *One-nat-def substitution-ops.subst-set-def card-eq-0-iff card-le-Suc0-iff-eq dual-order.eq-iff imageE le-Suc-eq*)
qed

lemma *is-unifier-singleton*[*simp*]: *is-unifier* υ $\{x\}$
by (*simp add*: *is-unifier-iff-if-finite*)

lemma *is-unifier-set-insert-singleton*[*simp*]:
is-unifier-set σ (*insert* $\{x\}$ *XX*) $\longleftrightarrow$ *is-unifier-set* σ *XX*
by (*simp add*: *is-unifier-set-def*)

lemma *is-mgu-insert-singleton*[*simp*]: *is-mgu* μ (*insert* $\{x\}$ *XX*) $\longleftrightarrow$ *is-mgu* μ *XX*
by (*simp add*: *is-mgu-def*)

lemma *is-imgu-insert-singleton*[*simp*]: *is-imgu* μ (*insert* $\{x\}$ *XX*) $\longleftrightarrow$ *is-imgu* μ *XX*
by (*simp add*: *is-imgu-def*)

lemma *subst-set-empty*[*simp*]: *subst-set* $\{\}$ $\sigma = \{\}$
by (*simp only*: *subst-set-def image-empty*)

lemma *subst-set-insert*[*simp*]: *subst-set* (*insert x X*) $\sigma =$ *insert* $(x \cdot \sigma)$ (*subst-set* X σ)
by (*simp only*: *subst-set-def image-insert*)

lemma *subst-set-union*[*simp*]: *subst-set* $(X1 \cup X2)$ $\sigma =$ *subst-set* $X1$ $\sigma \cup$ *subst-set* $X2$ σ
by (*simp only*: *subst-set-def image-Un*)

lemma *subst-list-Nil*[*simp*]: *subst-list* [] σ = []
by (*simp only*: *subst-list-def list.map*)

lemma *subst-list-insert*[*simp*]: *subst-list* (*x* # *xs*) $\sigma = (x \cdot \sigma)$ # (*subst-list xs* σ)
by (*simp only*: *subst-list-def list.map*)

lemma *subst-list-append*[*simp*]: *subst-list* $(xs_1 @ xs_2)$ $\sigma =$ *subst-list* xs_1 σ @ *subst-list* xs_2 σ
by (*simp only*: *subst-list-def map-append*)

lemma *is-unifier-set-union*:
is-unifier-set υ $(XX_1 \cup XX_2) \longleftrightarrow$ *is-unifier-set* υ $XX_1 \land$ *is-unifier-set* υ XX_2
by (*auto simp add*: *is-unifier-set-def*)

lemma *is-unifier-subset*: *is-unifier* υ $A \Longrightarrow$ *finite* $A \Longrightarrow B \subseteq A \Longrightarrow$ *is-unifier* υ B
by (*smt* (*verit*, *best*) *card-mono dual-order.trans finite-imageI image-mono is-unifier-def subst-set-def*)

lemma *is-ground-set-subset*: *is-ground-set* $A \Longrightarrow B \subseteq A \Longrightarrow$ *is-ground-set* B
by (*auto simp*: *is-ground-set-def*)

lemma *is-ground-set-ground-instances*[*simp*]: *is-ground-set* (*ground-instances* x)
by (*simp add*: *ground-instances-def is-ground-set-def*)

lemma *is-ground-set-ground-instances-set*[*simp*]: *is-ground-set* (*ground-instances-set* x)
by (*simp add*: *ground-instances-set-def is-ground-set-def*)

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*
using *comp-subst.lifting-monoid-action-to-set* **unfolding** *subst-set-def* **.**

sublocale *comp-subst-list*: *right-monoid-action comp-subst id-subst subst-list*
using *comp-subst.lifting-monoid-action-to-list* **unfolding** *subst-list-def* **.**

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*
  by (*simp add*: *is-renaming-def*)

lemma *is-unifier-id-subst-empty*[*simp*]: *is-unifier id-subst* {}
  by (*simp add*: *is-unifier-def*)

lemma *is-unifier-set-id-subst-empty*[*simp*]: *is-unifier-set id-subst* {}
  by (*simp add*: *is-unifier-set-def*)

lemma *is-mgu-id-subst-empty*[*simp*]: *is-mgu id-subst* {}
  by (*simp add*: *is-mgu-def*)

lemma *is-imgu-id-subst-empty*[*simp*]: *is-imgu id-subst* {}
  by (*simp add*: *is-imgu-def*)

lemma *is-idem-id-subst*[*simp*]: *is-idem id-subst*
  by (*simp add*: *is-idem-def*)

lemma *is-unifier-id-subst*: *is-unifier id-subst X* ⟷ *card X* ≤ *1*
  by (*simp add*: *is-unifier-def*)

lemma *is-unifier-set-id-subst*: *is-unifier-set id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
  by (*simp add*: *is-unifier-set-def is-unifier-id-subst*)

lemma *is-mgu-id-subst*: *is-mgu id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
  by (*simp add*: *is-mgu-def is-unifier-set-id-subst*)

lemma *is-imgu-id-subst*: *is-imgu id-subst XX* ⟷ (∀ *X* ∈ *XX*. *card X* ≤ *1*)
  by (*simp add*: *is-imgu-def is-unifier-set-id-subst*)

6.3 Generalization

sublocale *generalizes*: *preorder generalizes strictly-generalizes*
proof *unfold-locales*
  show ⋀*x y*. *strictly-generalizes x y* = (*generalizes x y* ∧ ¬ *generalizes y x*)
    unfolding *strictly-generalizes-def generalizes-def* **by** *blast*
next
  show ⋀*x*. *generalizes x x*
    unfolding *generalizes-def* **using** *subst-id-subst* **by** *metis*
next
  show ⋀*x y z*. *generalizes x y* ⟹ *generalizes y z* ⟹ *generalizes x z*
    unfolding *generalizes-def* **using** *subst-comp-subst* **by** *metis*
qed

6.4 Substituting on Ground Expressions

lemma *subst-ident-if-ground*[*simp*]: *is-ground* $x \Longrightarrow x \cdot \sigma = x$
using *all-subst-ident-if-ground* **by** *simp*

lemma *subst-set-ident-if-ground*[*simp*]: *is-ground-set* $X \Longrightarrow$ *subst-set* $X\ \sigma = X$
unfolding *is-ground-set-def subst-set-def* **by** *simp*

6.5 Instances of Ground Expressions

lemma *instances-ident-if-ground*[*simp*]: *is-ground* $x \Longrightarrow$ *instances* $x = \{x\}$
unfolding *instances-def generalizes-def* **by** *simp*

lemma *instances-set-ident-if-ground*[*simp*]: *is-ground-set* $X \Longrightarrow$ *instances-set* $X = X$
unfolding *instances-set-def is-ground-set-def* **by** *simp*

lemma *ground-instances-ident-if-ground*[*simp*]: *is-ground* $x \Longrightarrow$ *ground-instances* $x = \{x\}$
unfolding *ground-instances-def* **by** *auto*

lemma *ground-instances-set-ident-if-ground*[*simp*]: *is-ground-set* $X \Longrightarrow$ *ground-instances-set* $X = X$
unfolding *is-ground-set-def ground-instances-set-eq-Union-ground-instances* **by** *simp*

6.6 Unifier of Ground Expressions

lemma *ground-eq-ground-if-unifiable*:
assumes *is-unifier* $\upsilon\ \{t_1,\ t_2\}$ **and** *is-ground* t_1 **and** *is-ground* t_2
shows $t_1 = t_2$
using *assms* **by** (*simp add*: *card-Suc-eq is-unifier-def le-Suc-eq subst-set-def*)

lemma *ball-eq-constant-if-unifier*:
assumes *finite* X **and** $x \in X$ **and** *is-unifier* $\upsilon\ X$ **and** *is-ground-set* X
shows $\forall\, y \in X.\ y = x$
using *assms*
proof (*induction X rule*: *finite-induct*)
case *empty*
show *?case* **by** *simp*
next
case (*insert z F*)
then show *?case*
by (*metis is-ground-set-def finite.insertI is-unifier-iff-if-finite subst-ident-if-ground*)
qed

lemma *subst-mgu-eq-subst-mgu*:
assumes *is-mgu* $\mu\ \{\{t_1,\ t_2\}\}$
shows $t_1 \cdot \mu = t_2 \cdot \mu$
using *assms is-unifier-iff-if-finite*[*of* $\{t_1,\ t_2\}$]

unfolding *is-mgu-def is-unifier-set-def*
by *blast*

lemma *subst-imgu-eq-subst-imgu*:
assumes *is-imgu* μ $\{\{t_1,\ t_2\}\}$
shows $t_1 \cdot \mu = t_2 \cdot \mu$
using *assms is-unifier-iff-if-finite*[*of* $\{t_1,\ t_2\}$]
unfolding *is-imgu-def is-unifier-set-def*
by *blast*

6.7 Ground Substitutions

lemma *is-ground-subst-comp-left*: *is-ground-subst* $\sigma \Longrightarrow$ *is-ground-subst* $(\sigma \odot \tau)$
by (*simp add*: *is-ground-subst-def*)

lemma *is-ground-subst-comp-right*: *is-ground-subst* $\tau \Longrightarrow$ *is-ground-subst* $(\sigma \odot \tau)$
by (*simp add*: *is-ground-subst-def*)

lemma *is-ground-subst-is-ground*:
assumes *is-ground-subst* γ
shows *is-ground* $(t \cdot \gamma)$
using *assms is-ground-subst-def* **by** *blast*

6.8 IMGU is Idempotent and an MGU

lemma *is-imgu-iff-is-idem-and-is-mgu*: *is-imgu* μ *XX* $\longleftrightarrow$ *is-idem* μ $\wedge$ *is-mgu* μ *XX*
by (*auto simp add*: *is-imgu-def is-idem-def is-mgu-def simp flip*: *comp-subst.assoc*)

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 −
have $x \cdot \mu \cdot \upsilon = x \cdot (\mu \odot \upsilon)$
by *simp*

also have $\ldots = x \cdot \upsilon$
using *imgu unif* **by** (*simp add*: *is-imgu-def is-unifier-set-def*)

finally show *?thesis* **.**
qed

6.10 Groundings Idempotence

lemma *image-ground-instances-ground-instances*:
ground-instances ‘ *ground-instances* $x = (\lambda x.\ \{x\})$ ‘ *ground-instances* x
proof (*rule image-cong*)
show $\bigwedge x_{\mathcal{G}}.\ x_{\mathcal{G}} \in$ *ground-instances* $x \Longrightarrow$ *ground-instances* $x_{\mathcal{G}} = \{x_{\mathcal{G}}\}$

using *ground-instances-ident-if-ground ground-instances-def* **by** *auto*
qed *simp*

lemma *grounding-of-set-grounding-of-set-idem*[*simp*]:
ground-instances-set (*ground-instances-set X*) = *ground-instances-set X*
unfolding *ground-instances-set-eq-Union-ground-instances UN-UN-flatten*
unfolding *image-ground-instances-ground-instances*
by *simp*

6.11 Instances of Substitution

lemma *instances-subst*:
instances $(x \cdot \sigma) \subseteq$ *instances* x
proof (*rule subsetI*)
fix x_σ **assume** $x_\sigma \in$ *instances* $(x \cdot \sigma)$
thus $x_\sigma \in$ *instances* x
by (*metis CollectD CollectI generalizes-def instances-def subst-comp-subst*)
qed

lemma *instances-set-subst-set*:
instances-set (*subst-set X* σ) $\subseteq$ *instances-set X*
unfolding *instances-set-def subst-set-def*
using *instances-subst* **by** *auto*

lemma *ground-instances-subst*:
ground-instances $(x \cdot \sigma) \subseteq$ *ground-instances* x
unfolding *ground-instances-def*
using *instances-subst* **by** *auto*

lemma *ground-instances-set-subst-set*:
ground-instances-set (*subst-set X* σ) $\subseteq$ *ground-instances-set X*
unfolding *ground-instances-set-def*
using *instances-set-subst-set* **by** *auto*

6.12 Instances of Renamed Expressions

lemma *instances-subst-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *instances* $(x \cdot \varrho) =$ *instances* x
by (*metis instances-subst is-renaming-def subset-antisym subst-comp-subst subst-id-subst*)

lemma *instances-set-subst-set-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *instances-set* (*subst-set X* ϱ) = *instances-set X*
by (*simp add*: *instances-set-def subst-set-def*)

lemma *ground-instances-subst-ident-if-renaming*[*simp*]:
is-renaming $\varrho \Longrightarrow$ *ground-instances* $(x \cdot \varrho) =$ *ground-instances* x
by (*simp add*: *ground-instances-def*)

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*

by (*simp add*: *ground-instances-set-def*)

end

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

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

lemma *is-ground-iff*: *is-ground-trm* (*t* · *γ*) ⟷ (∀ *x* ∈ *vars-term t*. *is-ground-trm* (*γ x*))
by (*induction t*) *simp-all*

lemma *is-ground-trm-iff-ident-forall-subst*: *is-ground-trm t* ⟷ (∀ *σ*. *t* · *σ* = *t*)
proof(*induction t*)
case *Var*
then show *?case*
by *auto*
next
case *Fun*

moreover have ⋀*xs x σ*. ∀ *σ*. *map* (*λs*. *s* · *σ*) *xs* = *xs* ⟹ *x* ∈ *set xs* ⟹ *x* · *σ* = *x*
by (*metis list.map-ident map-eq-conv*)

ultimately show *?case*
by (*auto simp*: *map-idI*)
qed

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 *unfold-locales*
show ⋀*x*. *x* · *Var* = *x*
by *simp*
next
show ⋀*x σ τ*. *x* · *σ* $\circ_s$ *τ* = *x* · *σ* · *τ*
by *simp*
next
show ⋀*x*. *is-ground-trm x* ⟹ ∀ *σ*. *x* · *σ* = *x*
using *is-ground-trm-iff-ident-forall-subst* **..**
qed

lemma *term-subst-is-unifier-iff-unifiers*:
  assumes *finite X*
  shows *term-subst.is-unifier* μ *X* $\longleftrightarrow$ $\mu \in$ *unifiers* $(X \times X)$
  unfolding *term-subst.is-unifier-iff-if-finite*[*OF assms*] *unifiers-def*
  by *simp*

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* $(\bigcup X \in XX.\ X \times X)$
  using *term-subst-is-unifier-iff-unifiers assms*
  unfolding *term-subst.is-unifier-set-def unifiers-def*
  by *fast*

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 $(\bigcup X \in XX.\ X \times X)$
  using *term-subst-is-unifier-set-iff-unifiers*[*OF assms*]
  unfolding *term-subst.is-imgu-def is-imgu-def*
  by *auto*

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−
  have *is-imgu*: *is-imgu* μ $(\bigcup X \in XX.\ X \times X)$
    using *term-subst-is-imgu-iff-is-imgu*[*of XX*] *assms*
    by *simp*

  have *finite-prod*: *finite* $(\bigcup X \in XX.\ X \times X)$
    using *assms*
    by *blast*

  have $(\bigcup e \in \bigcup X \in XX.\ X \times X.$ *vars-term* (*fst e*) $\cup$ *vars-term* (*snd e*)$)$ = $(\bigcup t \in \bigcup XX.$ *vars-term t*$)$
    by *fastforce*

  then show *?thesis*
    using *imgu-range-vars-subset*[*OF is-imgu finite-prod*]
    by *argo*
qed

lemma *term-subst-is-renaming-iff*:
  term-subst.is-renaming ϱ $\longleftrightarrow$ *inj* ϱ $\wedge$ $(\forall x.$ *is-Var* $(\varrho\ x))$
proof (*rule iffI*)
  show *term-subst.is-renaming* ϱ $\Longrightarrow$ *inj* ϱ $\wedge$ $(\forall x.$ *is-Var* $(\varrho\ x))$
    unfolding *term-subst.is-renaming-def subst-compose-def inj-def*
    by (*metis term.sel*(*1*) *is-VarI subst-apply-eq-Var*)
next
  show *inj* ϱ $\wedge$ $(\forall x.$ *is-Var* $(\varrho\ x))$ $\Longrightarrow$ *term-subst.is-renaming* ϱ

unfolding *term-subst.is-renaming-def*
using *ex-inverse-of-renaming* **by** *metis*
qed

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 (*rule iffI*)
assume *term-subst.is-renaming* ϱ
hence *inj* ϱ **and** *all-Var*: $\forall x.\ is\text{-}Var\ (\varrho\ x)$
unfolding *term-subst-is-renaming-iff* **by** *simp-all*
from *all-Var* **obtain** *f* **where** $\forall x.\ \varrho\ x = Var\ (f\ x)$
by (*metis comp-apply term.collapse*(*1*))
hence $\varrho = Var \circ f$
using ‹$\forall x.\ \varrho\ x = Var\ (f\ x)$›
by (*intro ext*) *simp*
moreover have *inj f*
using ‹*inj* ϱ› **unfolding** ‹$\varrho = Var \circ f$›
using *inj-on-imageI2* **by** *metis*
ultimately show $\exists f.\ inj\ f \wedge \varrho = Var \circ f$
by *metis*
next
show $\exists f.\ inj\ f \wedge \varrho = Var \circ f \Longrightarrow$ *term-subst.is-renaming* ϱ
unfolding *term-subst-is-renaming-iff comp-apply inj-def*
by *auto*
qed

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 $t_1 = t_2$
using *assms*
using *term-subst.ground-eq-ground-if-unifiable*
by (*metis insertCI term-subst.is-imgu-def term-subst.is-unifier-set-def*)

lemma *the-mgu-is-unifier*:
assumes *term* · *the-mgu term term′* = *term′* · *the-mgu term term′*
shows *term-subst.is-unifier* (*the-mgu term term′*) {*term*, *term′*}
using *assms*
unfolding *term-subst.is-unifier-def the-mgu-def*
by *simp*

lemma *imgu-exists-extendable*:
fixes υ :: (*′f*, *′v*) *subst*
assumes *term* · υ = *term′* · υ *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
have *finite*: *finite* {*term*, *term′*}
by *simp*

have *term-subst.is-unifier-set* (*the-mgu term term′*) {{*term*, *term′*}}
unfolding *term-subst.is-unifier-set-def*
using *the-mgu-is-unifier*[*OF the-mgu*[*OF assms*(*1*), *THEN conjunct1*]]
by *simp*

moreover have
$\bigwedge\sigma$. *term-subst.is-unifier-set* σ {{*term*, *term′*}} $\Longrightarrow$ σ = *the-mgu term term′* $\circ_s$ σ
unfolding *term-subst.is-unifier-set-def*
using *term-subst.is-unifier-iff-if-finite*[*OF finite*] *the-mgu*
by *blast*

ultimately have *is-imgu*: *term-subst.is-imgu* (*the-mgu term term′*) {{*term*, *term′*}}
unfolding *term-subst.is-imgu-def*
by *metis*

show υ = (*the-mgu term term′*) $\circ_s$ υ
using *the-mgu*[*OF assms*(*1*)]
by *blast*

show *term-subst.is-imgu* (*the-mgu term term′*) {{*term*, *term′*}}
using *is-imgu*
by *blast*

show *P term term′* (*the-mgu term term′*)
using *assms*(*2*)**.**
qed

lemma *imgu-exists*:
fixes υ :: (*′f*, *′v*) *subst*
assumes *term* · υ = *term′* · υ
obtains μ :: (*′f*, *′v*) *subst*
where υ = μ $\circ_s$ υ *term-subst.is-imgu* μ {{*term*, *term′*}}
using *imgu-exists-extendable*[*OF assms*, *of* (λ- - -. *True*)]
by *auto*

lemma *is-renaming-if-term-subst-is-renaming*:
assumes *term-subst.is-renaming* ϱ
shows *is-renaming* ϱ
using *assms*
by (*simp add*: *inj-on-def is-renaming-def term-subst-is-renaming-iff*)

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 (*rule equivpI*)
show *reflp* ($\doteq$)
using *reflpI subst-refl* **by** *metis*
next
show *symp* ($\doteq$)
using *sympI subst-sym* **by** *metis*
next
show *transp* ($\doteq$)
using *transpI subst-trans* **by** *metis*
qed

lift-definition *subst-comp* :: *'a subst* ⇒ *'a subst* ⇒ *'a subst* (**infixl** ⊙ *67*)
is *Unification.comp*
using *Unification.subst-cong* **.**

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

global-interpretation *subst-comp*: *monoid subst-comp subst-id*
proof *unfold-locales*
show ⋀*a b c*. *a* ⊙ *b* ⊙ *c* = *a* ⊙ (*b* ⊙ *c*)
by (*smt* (*verit*, *del-insts*) *Quotient3-abs-rep Quotient3-subst Unification.comp-assoc subst.abs-eq-iff subst-comp.abs-eq*)
next
show ⋀*a*. *subst-id* ⊙ *a* = *a*
by (*metis Quotient3-abs-rep Quotient3-subst comp.simps*(*1*) *subst-comp.abs-eq subst-id-def*)
next
show ⋀*a*. *a* ⊙ *subst-id* = *a*
by (*metis Quotient3-abs-rep Quotient3-subst comp-Nil subst-comp.abs-eq subst-id-def*)
qed

lift-definition *subst-apply* :: *'a trm* ⇒ *'a subst* ⇒ *'a trm*
is *Unification.subst*
using *Unification.subst-eq-dest* **.**

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 *unfold-locales*

show $\bigwedge x\ a\ b.\ \mathit{subst\text{-}apply}\ x\ (a \odot b) = \mathit{subst\text{-}apply}\ (\mathit{subst\text{-}apply}\ x\ a)\ b$
by (*metis map-fun-apply subst-apply.abs-eq subst-apply.rep-eq subst-comp subst-comp-def*)
next
show $\bigwedge x.\ \mathit{subst\text{-}apply}\ x\ \mathit{subst\text{-}id} = x$
by (*simp add*: *subst-apply.abs-eq subst-id-def*)
next
show $\bigwedge x.\ \mathit{is\text{-}ground\text{-}trm}\ x \Longrightarrow \forall\,\sigma.\ \mathit{subst\text{-}apply}\ x\ \sigma = x$
by (*metis agreement empty-iff subst-Nil subst-apply.rep-eq*)
qed

end