Abstract Substituions as Monoid Actions

Martin Desharnais

March 17, 2025

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 *Monoid-Action*
  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* (∗) **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 Monoid

definition (**in** *monoid*) *is-left-invertible* **where**
  is-left-invertible $a \longleftrightarrow (\exists\ a\text{-}inv.\ a\text{-}inv * a = \mathbf{1})$

definition (**in** *monoid*) *is-right-invertible* **where**
  is-right-invertible $a \longleftrightarrow (\exists\ a\text{-}inv.\ a * a\text{-}inv = \mathbf{1})$

definition (**in** *monoid*) *left-inverse* **where**
  is-left-invertible $a \Longrightarrow$ *left-inverse* $a = (SOME\ a\text{-}inv.\ a\text{-}inv * a = \mathbf{1})$

definition (**in** *monoid*) *right-inverse* **where**
  is-right-invertible $a \Longrightarrow$ *right-inverse* $a = (SOME\ a\text{-}inv.\ a * a\text{-}inv = \mathbf{1})$

lemma (**in** *monoid*) *comp-left-inverse* [*simp*]:
  is-left-invertible $a \Longrightarrow$ *left-inverse* $a * a = \mathbf{1}$
  by (*auto simp*: *is-left-invertible-def left-inverse-def intro*: *someI-ex*)

lemma (**in** *monoid*) *comp-right-inverse* [*simp*]:
is-right-invertible $a \Longrightarrow a * \textit{right-inverse}\ a = \mathbf{1}$
by (*auto simp*: *is-right-invertible-def right-inverse-def intro*: *someI-ex*)

lemma (**in** *monoid*) *neutral-is-left-invertible* [*simp*]:
is-left-invertible **1**
by (*simp add*: *is-left-invertible-def*)

lemma (**in** *monoid*) *neutral-is-right-invertible* [*simp*]:
is-right-invertible **1**
by (*simp add*: *is-right-invertible-def*)

3 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** ‹·› *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** ‹·› *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 *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 $(*)$ $(\lambda X\ a.\ (\lambda x.\ \mathit{action}\ x\ a)\ `\ X)$
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 $(*)$ $(\lambda xs\ a.\ \mathit{map}\ (\lambda x.\ \mathit{action}\ x\ a)\ xs)$
proof *unfold-locales*
show $\bigwedge x\ a\ b.\ \mathit{map}\ (\lambda x.\ x \cdot (a * b))\ x = \mathit{map}\ (\lambda x.\ x \cdot b)\ (\mathit{map}\ (\lambda x.\ x \cdot a)\ x)$
by (*simp add*: *image-comp*)
qed

4 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* ⊆ *right-semigroup-action*
by *unfold-locales* (*fact monoid-action-compatibility*)

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

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

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

5 Group Action

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

sublocale *right-group-action* ⊆ *right-monoid-action*
  by *unfold-locales* (*fact group-action-compatibility group-action-neutral*)+

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

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

end
theory *Substitution*
  imports *Monoid-Action*
begin

abbreviation *set-prod* **where**

set-prod ≡ $\lambda(t,\ t').\ \{t,\ t'\}$

6 Assumption-free Substitution

locale *substitution-ops* =
  fixes
    subst :: $'x \Rightarrow 's \Rightarrow 'x$ (**infixl** ‹·› *67*) **and**
    id-subst :: $'s$ **and**
    comp-subst :: $'s \Rightarrow 's \Rightarrow 's$ (**infixl** ‹⊙› *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 $x\ y \longleftrightarrow generalizes\ x\ y \land \neg\ generalizes\ y\ x$

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

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 $x = \{x_{\mathcal{G}} \in instances\ x.\ is\text{-}ground\ x_{\mathcal{G}}\}$

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

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*)

lemma *mem-ground-instancesE*[*elim*]:
fixes $x\ x_G$ :: $'x$
assumes $x_G \in$ *ground-instances* x
obtains γ :: $'s$ **where** $x_G = x \cdot \gamma$ **and** *is-ground* $(x \cdot \gamma)$
using *assms*
unfolding *ground-instances-eq-Collect-subst-grounding mem-Collect-eq*
by *iprover*

lemma *mem-ground-instances-setE*[*elim*]:
fixes x_G :: $'x$ **and** X :: $'x$ *set*
assumes $x_G \in$ *ground-instances-set* X
obtains x :: $'x$ **and** γ :: $'s$ **where** $x \in X$ **and** $x_G = x \cdot \gamma$ **and** *is-ground* $(x \cdot \gamma)$
using *assms*
unfolding *ground-instances-set-eq-Union-ground-instances*
by *blast*

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 $XX \wedge (\forall \upsilon.$ *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 $XX \wedge (\forall \tau.$ *is-unifier-set* τ $XX \longrightarrow \mu \odot \tau = \tau)$

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-empty*[*simp*]:
is-unifier-set σ $\{\}$
by (*simp add*: *is-unifier-set-def*)

lemma *is-unifier-set-insert*:
is-unifier-set σ (*insert X XX*) $\longleftrightarrow$ *is-unifier* σ *X* $\wedge$ *is-unifier-set* σ *XX*
by (*simp add*: *is-unifier-set-def*)

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* $\{\}$ σ = $\{\}$
by (*simp only*: *subst-set-def image-empty*)

lemma *subst-set-insert*[*simp*]: *subst-set* (*insert x X*) σ = *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*) σ = *subst-set X1* σ $\cup$ *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*) σ = ($x \cdot \sigma$) # (*subst-list xs* σ)
by (*simp only*: *subst-list-def list.map*)

lemma *subst-list-append*[*simp*]: *subst-list* (xs_1 @ xs_2) σ = *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 $XX_1 \wedge$ *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

7 Basic Substitution

locale *substitution-monoid* = *monoid comp-subst id-subst*
for
comp-subst :: $'s \Rightarrow 's \Rightarrow 's$ **and**
id-subst :: $'s$
begin

abbreviation *is-renaming* **where**
is-renaming $\equiv$ *is-right-invertible*

lemmas *is-renaming-def* = *is-right-invertible-def*

abbreviation *renaming-inverse* **where**
renaming-inverse $\equiv$ *right-inverse*

lemmas *renaming-inverse-def* = *right-inverse-def*

lemmas *is-renaming-id-subst* = *neutral-is-right-invertible*

definition *is-idem* :: $'s \Rightarrow$ *bool* **where**
is-idem $a \longleftrightarrow$ *comp-subst* a $a = a$

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

end

locale *substitution* =
substitution-monoid comp-subst id-subst +
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** ‹⊙› *70*) **and**
id-subst :: $'s$ **and**
subst :: $'x \Rightarrow 's \Rightarrow 'x$ (**infixl** ‹·› *69*) **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* **.**

7.1 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*

7.2 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-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-unifier-id-subst*: *is-unifier id-subst* $X \longleftrightarrow$ *card* $X \leq 1$
by (*simp add*: *is-unifier-def*)

lemma *is-unifier-set-id-subst*: *is-unifier-set id-subst* $XX \longleftrightarrow (\forall X \in XX.$ *card* $X \leq 1)$

by (*simp add*: *is-unifier-set-def is-unifier-id-subst*)

lemma *is-mgu-id-subst*: *is-mgu id-subst XX* $\longleftrightarrow$ ($\forall$ *X* $\in$ *XX*. *card X* $\leq$ *1*)
by (*simp add*: *is-mgu-def is-unifier-set-id-subst*)

lemma *is-imgu-id-subst*: *is-imgu id-subst XX* $\longleftrightarrow$ ($\forall$ *X* $\in$ *XX*. *card X* $\leq$ *1*)
by (*simp add*: *is-imgu-def is-unifier-set-id-subst*)

7.3 Generalization

sublocale *generalizes*: *preorder generalizes strictly-generalizes*
proof *unfold-locales*
show $\bigwedge$*x y*. *strictly-generalizes x y* = (*generalizes x y* $\wedge$ $\neg$ *generalizes y x*)
unfolding *strictly-generalizes-def generalizes-def* **by** *blast*
next
show $\bigwedge$*x*. *generalizes x x*
unfolding *generalizes-def* **using** *subst-id-subst* **by** *metis*
next
show $\bigwedge$*x y z*. *generalizes x y* $\Longrightarrow$ *generalizes y z* $\Longrightarrow$ *generalizes x z*
unfolding *generalizes-def* **using** *subst-comp-subst* **by** *metis*
qed

lemma *generalizes-antisym-if*:
assumes $\bigwedge\sigma_1\ \sigma_2\ x.\ x \cdot (\sigma_1 \odot \sigma_2) = x \Longrightarrow x \cdot \sigma_1 = x$
shows $\bigwedge$*x y*. *generalizes x y* $\Longrightarrow$ *generalizes y x* $\Longrightarrow$ *x* = *y*
using *assms*
by (*metis generalizes-def subst-comp-subst*)

lemma *order-generalizes-if*:
assumes $\bigwedge\sigma_1\ \sigma_2\ x.\ x \cdot (\sigma_1 \odot \sigma_2) = x \Longrightarrow x \cdot \sigma_1 = x$
shows *class.order generalizes strictly-generalizes*
proof *unfold-locales*
show $\bigwedge$*x y*. *generalizes x y* $\Longrightarrow$ *generalizes y x* $\Longrightarrow$ *x* = *y*
using *generalizes-antisym-if assms* **by** *iprover*
qed

7.4 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* σ = *X*
unfolding *is-ground-set-def subst-set-def* **by** *simp*

7.5 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*

7.6 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 $t_1 = t_2$
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* υ 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 *is-mgu-unifies*:
assumes *is-mgu* μ XX $\forall X \in XX.$ *finite* X
shows $\forall X \in XX.\ \forall t \in X.\ \forall t' \in X.\ t \cdot \mu = t' \cdot \mu$
using *assms is-unifier-iff-if-finite*
unfolding *is-mgu-def is-unifier-set-def*
by *blast*

corollary *is-mgu-unifies-pair*:
assumes *is-mgu* μ $\{\{t, t'\}\}$
shows $t \cdot \mu = t' \cdot \mu$
using *is-mgu-unifies*[*OF assms*]
by (*metis finite.emptyI finite.insertI insertCI singletonD*)

lemmas *subst-mgu-eq-subst-mgu* = *is-mgu-unifies-pair*

lemma *is-imgu-unifies*:

assumes *is-imgu* μ *XX* $\forall X \in XX$. *finite* X
shows $\forall X \in XX. \forall t \in X. \forall t' \in X. t \cdot \mu = t' \cdot \mu$
using *assms is-unifier-iff-if-finite*
unfolding *is-imgu-def is-unifier-set-def*
by *blast*

corollary *is-imgu-unifies-pair*:
assumes *is-imgu* μ $\{\{t, t'\}\}$
shows $t \cdot \mu = t' \cdot \mu$
using *is-imgu-unifies*[*OF assms*]
by (*metis finite.emptyI finite.insertI insertCI singletonD*)

lemmas *subst-imgu-eq-subst-imgu* = *is-imgu-unifies-pair*

7.7 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*

7.8 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*: *assoc*)

7.9 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

7.10 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*

7.11 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*

7.12 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 $t \equiv$ *vars-term* $t = \{\}$

lemma *is-ground-iff*: *is-ground-trm* $(t \cdot \gamma) \longleftrightarrow (\forall x \in$ *vars-term* t. *is-ground-trm* $(\gamma\ x))$
  by (*induction* t) *simp-all*

lemma *is-ground-trm-iff-ident-forall-subst*: *is-ground-trm* $t \longleftrightarrow (\forall \sigma.\ t \cdot \sigma = t)$
proof(*induction* t)
  case *Var*
  then show *?case*
    by *auto*
next
  case *Fun*

  moreover have $\bigwedge xs\ x\ \sigma.\ \forall \sigma.\ map\ (\lambda s.\ s \cdot \sigma)\ xs = xs \Longrightarrow x \in set\ xs \Longrightarrow x \cdot \sigma = x$
    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 $\bigwedge x.\ x \cdot Var = x$
    by *simp*
next

show $\bigwedge x\ \sigma\ \tau.\ x \cdot \sigma \circ_s \tau = x \cdot \sigma \cdot \tau$
by *simp*
next
show $\bigwedge x.$ *is-ground-trm* $x \Longrightarrow \forall \sigma.\ x \cdot \sigma = x$
using *is-ground-trm-iff-ident-forall-subst* **..**
qed

lemma *term-subst-is-unifier-iff-unifiers-Times*:
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-Union-Times*:
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-Times assms*
unfolding *term-subst.is-unifier-set-def unifiers-def*
by *fast*

lemma *term-subst-is-mgu-iff-is-mgu-Union-Times*:
assumes *fin*: $\forall X \in XX.$ *finite X*
shows *term-subst.is-mgu* μ $XX \longleftrightarrow$ *is-mgu* μ $(\bigcup X \in XX.\ X \times X)$
unfolding *term-subst.is-mgu-def is-mgu-def*
unfolding *term-subst-is-unifier-set-iff-unifiers-Union-Times*[*OF fin*]
by *auto*

lemma *term-subst-is-imgu-iff-is-imgu-Union-Times*:
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-Union-Times*[*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-Union-Times*[*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 $\varrho \longleftrightarrow$ *inj* $\varrho \wedge (\forall x.$ *is-Var* $(\varrho\ x))$
proof (*rule iffI*)
show *term-subst.is-renaming* $\varrho \Longrightarrow$ *inj* $\varrho \wedge (\forall x.$ *is-Var* $(\varrho\ x))$
unfolding *term-subst.is-renaming-def*
unfolding *subst-compose-def inj-def*
by (*metis subst-apply-eq-Var term.discI*(*1*) *term.inject*(*1*))
next
show *inj* $\varrho \wedge (\forall x.$ *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-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 *is-unifier-the-mgu*:
  assumes $t \cdot \textit{the-mgu}\ t\ t' = t' \cdot \textit{the-mgu}\ t\ t'$
  shows $\textit{term-subst.is-unifier}\ (\textit{the-mgu}\ t\ t')\ \{t,\ t'\}$
  using *assms*
  unfolding *term-subst.is-unifier-def the-mgu-def*
  by *simp*

lemma *obtains-imgu-from-unifier-and-the-mgu*:
  fixes υ :: $('f,\ 'v)$ *subst*
  assumes $t \cdot \upsilon = t' \cdot \upsilon$ $P\ t\ t'\ (\textit{Unification.the-mgu}\ t\ t')$
  obtains μ :: $('f,\ 'v)$ *subst*
  where $\upsilon = \mu \circ_s \upsilon$ $\textit{term-subst.is-imgu}\ \mu\ \{\{t,\ t'\}\}$ $P\ t\ t'\ \mu$
proof
  have *finite*: *finite* $\{t,\ t'\}$
    by *simp*

  have $\textit{term-subst.is-unifier-set}\ (\textit{the-mgu}\ t\ t')\ \{\{t,\ t'\}\}$
    unfolding *term-subst.is-unifier-set-def*
    using *is-unifier-the-mgu*[*OF the-mgu*[*OF assms*(*1*), *THEN conjunct1*]]
    by *simp*

  moreover have $\bigwedge\sigma.\ \textit{term-subst.is-unifier-set}\ \sigma\ \{\{t,\ t'\}\} \Longrightarrow \sigma = \textit{the-mgu}\ t\ t'$
$\circ_s\ \sigma$
    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*: $\textit{term-subst.is-imgu}\ (\textit{the-mgu}\ t\ t')\ \{\{t,\ t'\}\}$
    unfolding *term-subst.is-imgu-def*
    by *metis*

  show $\upsilon = (\textit{the-mgu}\ t\ t') \circ_s \upsilon$
    using *the-mgu*[*OF assms*(*1*)]
    by *blast*

  show $\textit{term-subst.is-imgu}\ (\textit{the-mgu}\ t\ t')\ \{\{t,\ t'\}\}$
    using *is-imgu*
    by *blast*

  show $P\ t\ t'\ (\textit{the-mgu}\ t\ t')$
    using *assms*(*2*)**.**
qed

lemma *obtains-imgu*:
  fixes υ :: $('f,\ 'v)$ *subst*
  assumes $t \cdot \upsilon = t' \cdot \upsilon$
  obtains μ :: $('f,\ 'v)$ *subst*
  where $\upsilon = \mu \circ_s \upsilon$ $\textit{term-subst.is-imgu}\ \mu\ \{\{t,\ t'\}\}$
  using *obtains-imgu-from-unifier-and-the-mgu*[*OF assms*, *of* (λ- - -. *True*)]

by *auto*

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

lemma *is-mgu-iff-term-subst-is-imgu-image-set-prod*:
fixes μ :: ('*f*, '*v*) *subst* **and** X :: ((('*f*, '*v*) *term* $\times$ ('*f*, '*v*) *term*) *set*
shows *Unifiers.is-imgu* μ $X \longleftrightarrow$ *term-subst.is-imgu* μ (*set-prod* ‘ X)
proof (*rule iffI*)
assume *is-imgu* μ X

moreover then have
$\forall e \in X.$ *fst* $e \cdot \mu =$ *snd* $e \cdot \mu$
$\forall \tau$ :: ('*f*, '*v*) *subst*. ($\forall e \in X.$ *fst* $e \cdot \tau =$ *snd* $e \cdot \tau$) $\longrightarrow \tau = \mu \circ_s \tau$
unfolding *is-imgu-def unifiers-def*
by *auto*

moreover then have
$\bigwedge\tau$ :: ('*f*, '*v*) *subst*. $\forall e \in X.$ $\forall t\ t'.$ $e = (t, t') \longrightarrow$ *card* $\{t \cdot \tau, t' \cdot \tau\} \leq$ *Suc 0*
$\Longrightarrow \mu \circ_s \tau = \tau$
by (*metis Suc-n-not-le-n card-1-singleton-iff card-Suc-eq insert-iff prod.collapse*)

ultimately show *term-subst.is-imgu* μ (*set-prod* ‘ X)
unfolding *term-subst.is-imgu-def term-subst.is-unifier-set-def term-subst.is-unifier-def*
by (*auto split*: *prod.splits*)
next
assume *is-imgu*: *term-subst.is-imgu* μ (*set-prod* ‘ X)

show *is-imgu* μ X
proof(*unfold is-imgu-def unifiers-def*, *intro conjI ballI*)

show $\mu \in \{\sigma.\ \forall e \in X.$ *fst* $e \cdot \sigma =$ *snd* $e \cdot \sigma\}$
using *term-subst.is-imgu-unifies*[*OF is-imgu*]
by *fastforce*
next
fix τ :: ('*f*, '*v*) *subst*
assume $\tau \in \{\sigma.\ \forall e \in X.$ *fst* $e \cdot \sigma =$ *snd* $e \cdot \sigma\}$

then have $\forall e \in X.$ *fst* $e \cdot \tau =$ *snd* $e \cdot \tau$
by *blast*

then show $\tau = \mu \circ_s \tau$
using *is-imgu*
unfolding *term-subst.is-imgu-def term-subst.is-unifier-set-def*
by (*smt* (*verit*, *del-insts*) *case-prod-conv empty-iff finite.emptyI finite.insertI*

```
image-iff
          insert-iff prod.collapse term-subst.is-unifier-iff-if-finite)
  qed
qed

lemma the-mgu-term-subst-is-imgu:
  fixes υ :: ('f, 'v) subst
  assumes s · υ = t · υ
  shows term-subst.is-imgu (Unification.the-mgu s t) {{s, t}}
  using the-mgu-is-imgu[OF assms]
  unfolding is-mgu-iff-term-subst-is-imgu-image-set-prod
  by simp

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 / (≐)
proof (rule equivpI)
  show reflp (≐)
    using reflpI subst-refl by metis
next
  show symp (≐)
    using sympI subst-sym by metis
next
  show transp (≐)
    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 $\bigwedge a.\ a \odot subst\text{-}id = a$
   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 $\bigwedge x\ a\ b.\ subst\text{-}apply\ x\ (a \odot b) = subst\text{-}apply\ (subst\text{-}apply\ x\ a)\ b$
   by (*metis map-fun-apply subst-apply.abs-eq subst-apply.rep-eq subst-comp subst-comp-def*)
next
  show $\bigwedge x.\ subst\text{-}apply\ x\ subst\text{-}id = x$
    by (*simp add*: *subst-apply.abs-eq subst-id-def*)
next
  show $\bigwedge x.\ is\text{-}ground\text{-}trm\ x \Longrightarrow \forall \sigma.\ subst\text{-}apply\ x\ \sigma = x$
    by (*metis agreement empty-iff subst-Nil subst-apply.rep-eq*)
qed

end
theory *Functional-Substitution*
  imports
    Substitution
    HOL−Library.FSet
begin

locale *functional-substitution* = *substitution* **where**
  subst = *subst* **and** *is-ground* = λ*expr*. *vars expr* = {}
  for
    subst :: *'expr* ⇒ (*'var* ⇒ *'base*) ⇒ *'expr* (**infixl** · *69*) **and**
    vars :: *'expr* ⇒ *'var set* +
  assumes
    subst-eq: $\bigwedge expr\ \sigma\ \tau.\ (\bigwedge x.\ x \in vars\ expr \Longrightarrow \sigma\ x = \tau\ x) \Longrightarrow expr \cdot \sigma = expr \cdot$
τ
begin

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

definition *vars-set* :: *'expr set* ⇒ *'var set* **where**
  vars-set exprs ≡ $\bigcup$ *expr* ∈ *exprs*. *vars expr*

lemma *subst-reduntant-upd* [*simp*]:
  assumes $var \notin vars\ expr$
  shows $expr \cdot \sigma(var := update) = expr \cdot \sigma$
  using *assms subst-eq*
  by *fastforce*

lemma *subst-reduntant-if* [*simp*]:
  assumes $vars\ expr \subseteq vars'$
  shows $expr \cdot (\lambda var.\ if\ var \in vars'\ then\ \sigma\ var\ else\ \sigma'\ var) = expr \cdot \sigma$
  using *assms*
  by (*smt* (*verit*, *best*) *subset-eq subst-eq*)

lemma *subst-reduntant-if'* [*simp*]:
  assumes $vars\ expr \cap vars' = \{\}$
  shows $expr \cdot (\lambda var.\ if\ var \in vars'\ then\ \sigma'\ var\ else\ \sigma\ var) = expr \cdot \sigma$
  using *assms subst-eq*
  unfolding *disjoint-iff*
  by *presburger*

lemma *subst-cannot-unground*:
  assumes $\neg is\text{-}ground\ (expr \cdot \sigma)$
  shows $\neg is\text{-}ground\ expr$
  using *assms* **by** *force*

definition *subst-domain* :: ('*var* ⇒ '*base*) ⇒ '*var set* **where**
  $subst\text{-}domain\ \sigma = \{x.\ \sigma\ x \neq id\text{-}subst\ x\}$

abbreviation *subst-range* :: ('*var* ⇒ '*base*) ⇒ '*base set* **where**
  $subst\text{-}range\ \sigma \equiv \sigma\ `\ subst\text{-}domain\ \sigma$

lemma *subst-inv*:
  assumes $\sigma \odot \sigma\text{-}inv = id\text{-}subst$
  shows $expr \cdot \sigma \cdot \sigma\text{-}inv = expr$
  using *assms*
  by (*metis subst-comp-subst subst-id-subst*)

definition *rename* **where**
  $is\text{-}renaming\ \varrho \Longrightarrow rename\ \varrho\ x \equiv SOME\ x'.\ \varrho\ x = id\text{-}subst\ x'$

end

locale *all-subst-ident-iff-ground* =
  functional-substitution +
  assumes
    $all\text{-}subst\text{-}ident\text{-}iff\text{-}ground: \bigwedge expr.\ is\text{-}ground\ expr \longleftrightarrow (\forall \sigma.\ subst\ expr\ \sigma = expr)$
and
    exists-non-ident-subst:
      $\bigwedge expr\ S.\ finite\ S \Longrightarrow \neg is\text{-}ground\ expr \Longrightarrow \exists \sigma.\ subst\ expr\ \sigma \neq expr \wedge subst\ expr\ \sigma \notin S$

locale *finite-variables* = *functional-substitution* **where** *vars* = *vars*
for *vars* :: *'expr* ⇒ *'var set* +
assumes *finite-vars* [*intro*]: ⋀*expr*. *finite* (*vars expr*)
begin

abbreviation *finite-vars* :: *'expr* ⇒ *'var fset* **where**
finite-vars expr ≡ *Abs-fset* (*vars expr*)

lemma *fset-finite-vars* [*simp*]: *fset* (*finite-vars expr*) = *vars expr*
using *Abs-fset-inverse finite-vars*
by *blast*

end

locale *renaming-variables* = *functional-substitution* +
assumes
is-renaming-iff: ⋀*ϱ*. *is-renaming ϱ* ⟷ *inj ϱ* ∧ (∀ *x*. ∃ *x'*. *ϱ x* = *id-subst x'*)
and
rename-variables: ⋀*expr ϱ*. *is-renaming ϱ* ⟹ *vars* (*expr* · *ϱ*) = *rename ϱ* ‘ (*vars expr*)
begin

lemma *renaming-range-id-subst*:
assumes *is-renaming ϱ*
shows *ϱ x* ∈ *range id-subst*
using *assms*
unfolding *is-renaming-iff*
by *auto*

lemma *obtain-renamed-variable*:
assumes *is-renaming ϱ*
obtains *x'* **where** *ϱ x* = *id-subst x'*
using *renaming-range-id-subst*[*OF assms*]
by *auto*

lemma *id-subst-rename* [*simp*]:
assumes *is-renaming ϱ*
shows *id-subst* (*rename ϱ x*) = *ϱ x*
unfolding *rename-def*[*OF assms*]
using *obtain-renamed-variable*[*OF assms*]
by (*metis* (*mono-tags*, *lifting*) *someI*)

lemma *rename-variables-id-subst*:
assumes *is-renaming ϱ*
shows *id-subst* ‘ *vars* (*expr* · *ϱ*) = *ϱ* ‘ (*vars expr*)
using *rename-variables*[*OF assms*] *id-subst-rename*[*OF assms*]
by (*metis* (*no-types*, *lifting*) *image-cong image-image*)

lemma *surj-inv-renaming*:
assumes *is-renaming* ϱ
shows *surj* ($\lambda x.$ *inv* ϱ (*id-subst* x))
using *assms inv-f-f*
unfolding *is-renaming-iff surj-def*
by *metis*

lemma *renaming-range*:
assumes *is-renaming* ϱ $x \in$ *vars* (*expr* $\cdot$ ϱ)
shows *id-subst* $x \in$ *range* ϱ
using *rename-variables-id-subst*[*OF assms*(*1*)] *assms*(*2*)
by *fastforce*

lemma *renaming-inv-into*:
assumes *is-renaming* ϱ $x \in$ *vars* (*expr* $\cdot$ ϱ)
shows ϱ (*inv* ϱ (*id-subst* x)) = *id-subst* x
using *f-inv-into-f*[*OF renaming-range*[*OF assms*]]**.**

lemma *inv-renaming*:
assumes *is-renaming* ϱ
shows *inv* ϱ (ϱ x) = x
using *assms*
unfolding *is-renaming-iff*
by *simp*

lemma *renaming-inv-in-vars*:
assumes *is-renaming* ϱ $x \in$ *vars* (*expr* $\cdot$ ϱ)
shows *inv* ϱ (*id-subst* x) $\in$ *vars expr*
using *assms rename-variables-id-subst*[*OF assms*(*1*)]
by (*metis image-eqI image-inv-f-f is-renaming-iff*)

end

locale *grounding* = *functional-substitution* **where** *vars* = *vars* **and** *id-subst* = *id-subst*
for *vars* :: *'expr* $\Rightarrow$ *'var set* **and** *id-subst* :: *'var* $\Rightarrow$ *'base* +
fixes *to-ground* :: *'expr* $\Rightarrow$ *'expr*$_G$ **and** *from-ground* :: *'expr*$_G$ $\Rightarrow$ *'expr*
assumes
range-from-ground-iff-is-ground: {*expr*. *is-ground expr*} = *range from-ground*
and
from-ground-inverse [*simp*]: $\bigwedge$*expr*$_G$. *to-ground* (*from-ground expr*$_G$) = *expr*$_G$
begin

definition *ground-instances'* ::*'expr* $\Rightarrow$ *'expr*$_G$ *set* **where**
ground-instances' expr = { *to-ground* (*expr* $\cdot$ γ) | γ. *is-ground* (*expr* $\cdot$ γ) }

lemma *ground-instances'-eq-ground-instances*:
ground-instances' expr = (*to-ground* ‘ *ground-instances expr*)
unfolding *ground-instances'-def ground-instances-def generalizes-def instances-def*

by *blast*

lemma *to-ground-from-ground-id* [*simp*]: *to-ground* ∘ *from-ground* = *id*
using *from-ground-inverse*
by *auto*

lemma *surj-to-ground*: *surj to-ground*
using *from-ground-inverse*
by (*metis surj-def*)

lemma *inj-from-ground*: *inj-on from-ground* $domain_G$
by (*metis from-ground-inverse inj-on-inverseI*)

lemma *inj-on-to-ground*: *inj-on to-ground* (*from-ground* ‘ $domain_G$)
unfolding *inj-on-def*
by *simp*

lemma *bij-betw-to-ground*: *bij-betw to-ground* (*from-ground* ‘ $domain_G$) $domain_G$
by (*smt* (*verit*, *best*) *bij-betwI′ from-ground-inverse image-iff*)

lemma *bij-betw-from-ground*: *bij-betw from-ground* $domain_G$ (*from-ground* ‘ $domain_G$)
by (*simp add*: *bij-betw-def inj-from-ground*)

lemma *ground-is-ground* [*simp*, *intro*]: *is-ground* (*from-ground* $expr_G$)
using *range-from-ground-iff-is-ground*
by *blast*

lemma *is-ground-iff-range-from-ground*: *is-ground expr* ⟷ *expr* ∈ *range from-ground*
using *range-from-ground-iff-is-ground*
by *auto*

lemma *to-ground-inverse* [*simp*]:
assumes *is-ground expr*
shows *from-ground* (*to-ground expr*) = *expr*
using *inj-on-to-ground from-ground-inverse is-ground-iff-range-from-ground assms*
unfolding *inj-on-def*
by *blast*

corollary *obtain-grounding*:
assumes *is-ground expr*
obtains $expr_G$ **where** *from-ground* $expr_G$ = *expr*
using *to-ground-inverse assms*
by *blast*

lemma *from-ground-eq* [*simp*]:
from-ground expr = *from-ground expr′* ⟷ *expr* = *expr′*
by (*metis from-ground-inverse*)

lemma *to-ground-eq* [*simp*]:
  assumes *is-ground expr is-ground expr′*
  shows *to-ground expr* = *to-ground expr′* ⟷ *expr* = *expr′*
  using *assms obtain-grounding*
  by *fastforce*

end

locale *base-functional-substitution* = *functional-substitution*
  where *id-subst* = *id-subst* **and** *vars* = *vars*
  for *id-subst* :: *′var* ⇒ *′expr* **and** *vars* :: *′expr* ⇒ *′var set* +
  assumes
    vars-subst-vars: ⋀*expr* ϱ. *vars* (*expr* · ϱ) = ⋃ (*vars* ‘ ϱ ‘ *vars expr*) **and**
    base-ground-exists: ∃ *expr*. *is-ground expr* **and**
    vars-id-subst: ⋀*x*. *vars* (*id-subst x*) = {*x*} **and**
    comp-subst-iff: ⋀σ σ′ *x*. (σ ⊙ σ′) *x* = σ *x* · σ′

locale *based-functional-substitution* =
 base: *base-functional-substitution* **where** *subst* = *base-subst* **and** *vars* = *base-vars*
+
 functional-substitution **where** *vars* = *vars*
for
 base-subst :: *′base* ⇒ (*′var* ⇒ *′base*) ⇒ *′base* **and**
 base-vars **and**
 vars :: *′expr* ⇒ *′var set* +
assumes
 ground-subst-iff-base-ground-subst [*simp*]: ⋀γ. *is-ground-subst* γ ⟷ *base.is-ground-subst*
γ **and**
 vars-subst: ⋀*expr* ϱ. *vars* (*expr* · ϱ) = ⋃ (*base-vars* ‘ ϱ ‘ *vars expr*)
begin

lemma *is-grounding-iff-vars-grounded*:
 is-ground (*expr* · γ) ⟷ (∀ *var* ∈ *vars expr*. *base.is-ground* (γ *var*))
  using *vars-subst*
  by *auto*

lemma *obtain-ground-subst*:
  obtains γ
  where *is-ground-subst* γ
  unfolding *ground-subst-iff-base-ground-subst base.is-ground-subst-def*
  using *base.base-ground-exists base.vars-subst-vars*
  by (*meson is-ground-subst-def is-grounding-iff-vars-grounded that*)

lemma *exists-ground-subst* [*intro*]: ∃ γ. *is-ground-subst* γ
  by (*metis obtain-ground-subst*)

lemma *ground-subst-extension*:
  assumes *is-ground* (*expr* · γ)

obtains γ'
where $expr \cdot \gamma = expr \cdot \gamma'$ **and** *is-ground-subst* γ'
using *obtain-ground-subst assms*
by (*metis all-subst-ident-if-ground is-ground-subst-comp-right subst-comp-subst*)

lemma *ground-subst-extension'*:
assumes *is-ground* $(expr \cdot \gamma)$
obtains γ'
where $expr \cdot \gamma = expr \cdot \gamma'$ **and** *base.is-ground-subst* γ'
using *ground-subst-extension assms*
by *auto*

lemma *ground-subst-update* [*simp*]:
assumes *base.is-ground update is-ground* $(expr \cdot \gamma)$
shows *is-ground* $(expr \cdot \gamma(var := update))$
using *assms is-grounding-iff-vars-grounded*
by *auto*

lemma *ground-exists*: $\exists$ *expr. is-ground expr*
using *base.base-ground-exists*
by (*meson is-grounding-iff-vars-grounded*)

lemma *variable-grounding*:
assumes *is-ground* $(expr \cdot \gamma)$ $var \in vars\ expr$
shows *base.is-ground* $(\gamma\ var)$
using *assms is-grounding-iff-vars-grounded*
by *blast*

definition *range-vars* :: $('var \Rightarrow 'base) \Rightarrow 'var\ set$ **where**
$range\text{-}vars\ \sigma = \bigcup(base\text{-}vars\ `\ subst\text{-}range\ \sigma)$

lemma *vars-subst-subset*: $vars\ (expr \cdot \sigma) \subseteq (vars\ expr - subst\text{-}domain\ \sigma) \cup range\text{-}vars\ \sigma$
unfolding *subst-domain-def range-vars-def vars-subst*
using *base.vars-id-subst*
by (*smt* (*verit*, *del-insts*) *Diff-iff UN-iff UnCI image-iff mem-Collect-eq singletonD subsetI*)

end

locale *variables-in-base-imgu* = *based-functional-substitution* +
assumes *variables-in-base-imgu*:
$\bigwedge expr\ \mu\ unifications.$
$base.is\text{-}imgu\ \mu\ unifications \Longrightarrow$
$finite\ unifications \Longrightarrow$
$\forall\ unification \in unifications.\ finite\ unification \Longrightarrow$
$vars\ (expr \cdot \mu) \subseteq vars\ expr \cup (\bigcup(base\text{-}vars\ `\ \bigcup\ unifications))$

context *base-functional-substitution*

begin

sublocale *based-functional-substitution*
  where *base-subst* = *subst* **and** *base-vars* = *vars*
  by *unfold-locales* (*simp-all add*: *vars-subst-vars*)

declare *ground-subst-iff-base-ground-subst* [*simp del*]

end

hide-fact *base-functional-substitution.base-ground-exists*
hide-fact *base-functional-substitution.vars-subst-vars*

end
theory *Functional-Substitution-Example*
  imports *Functional-Substitution Substitution-First-Order-Term*
begin

A selection of substitution properties for terms.

locale *term-subst-properties* =
  base-functional-substitution +
  finite-variables

global-interpretation *term-subst*: *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*
rewrites ⋀*t*. *term-subst.is-ground t* ⟷ *ground t*

"rewrites" enables us to use our own equivalent definitions.

proof *unfold-locales*
  fix *t* :: ('*f*, '*v*) *term* **and** *σ τ* :: ('*f*, '*v*) *subst*
  assume ⋀*x*. *x* ∈ *vars-term t* ⟹ *σ x* = *τ x*
  then show *t* · *σ* = *t* · *τ*
    by(*rule term-subst-eq*)
next
  fix *t* :: ('*f*, '*v*) *term*
  show *finite* (*vars-term t*)
    by *simp*
next
  fix *t* :: ('*f*, '*v*) *term* **and** *ϱ* :: ('*f*, '*v*) *subst*

  show *vars-term* (*t* · *ϱ*) = ⋃ (*vars-term* ‘ *ϱ* ‘ *vars-term t*)
    using *vars-term-subst*.
next
  show ∃ *t*. *vars-term t* = {}
    by(*rule exI*[*of* - *Fun* (*SOME f*. *True*) []]) *auto*
next
  fix *x* :: '*v*

```
  show vars-term (Var x) = {x}
    by simp
next
  fix σ σ' :: ('f, 'v) subst and x
  show (σ ∘s σ') x = σ x · σ'
    using subst-compose.
next
  fix t :: ('f, 'v) term
  show is-ground-trm t = ground t
    by (induction t) auto
qed
```

Examples of generated lemmas and definitions

```
thm
  term-subst.subst-reduntant-upd
  term-subst.subst-reduntant-if

  term-subst.subst-domain-def
  term-subst.range-vars-def

  term-subst.vars-subst-subset

end
theory Natural-Magma
  imports Main
begin

locale natural-magma =
  fixes
    to-set :: 'b ⇒ 'a set and
    plus :: 'b ⇒ 'b ⇒ 'b and
    wrap :: 'a ⇒ 'b and
    add
  defines ⋀a b. add a b ≡ plus (wrap a) b
  assumes
    to-set-plus [simp]: ⋀b b'. to-set (plus b b') = (to-set b) ∪ (to-set b') and
    to-set-wrap [simp]: ⋀a. to-set (wrap a) = {a}
begin

lemma to-set-add [simp]: to-set (add a b) = insert a (to-set b)
  using to-set-plus to-set-wrap add-def
  by simp

end

locale natural-magma-with-empty = natural-magma +
  fixes empty
  assumes to-set-empty [simp]: to-set empty = {}
```

end
theory *Natural-Functor*
  imports *Main*
begin

locale *natural-functor* =
  fixes
    map :: ($'a \Rightarrow 'a$) $\Rightarrow$ $'b \Rightarrow 'b$ **and**
    to-set :: $'b \Rightarrow 'a$ *set*
  assumes
    map-comp [*simp*]: $\bigwedge b\ f\ g.$ *map f* (*map g b*) = *map* ($\lambda x.$ *f* (*g x*)) *b* **and**
    map-ident [*simp*]: $\bigwedge b.$ *map* ($\lambda x.$ *x*) *b* = *b* **and**
    map-cong0 [*cong*]: $\bigwedge b\ f\ g.$ ($\bigwedge a.$ *a* $\in$ *to-set b* $\Longrightarrow$ *f a* = *g a*) $\Longrightarrow$ *map f b* = *map g b* **and**
    to-set-map [*simp*]: $\bigwedge b\ f.$ *to-set* (*map f b*) = *f* ‘ *to-set b* **and**
    exists-functor [*intro*]: $\bigwedge a.$ $\exists$ *b. a* $\in$ *to-set b*
begin

lemma *map-id* [*simp*]: *map id b* = *b*
  unfolding *id-def*
  by(*rule map-ident*)

lemma *map-cong* [*cong*]:
  assumes *b* = *b'* $\bigwedge a.$ *a* $\in$ *to-set b'* $\Longrightarrow$ *f a* = *g a*
  shows *map f b* = *map g b'*
  using *map-cong0 assms*
  by *blast*

end

locale *finite-natural-functor* = *natural-functor* +
  assumes *finite-to-set* [*intro*]: $\bigwedge b.$ *finite* (*to-set b*)

locale *natural-functor-conversion* =
  natural-functor +
  functor': *natural-functor* **where** *map* = *map'* **and** *to-set* = *to-set'*
  for *map'* :: ($'b \Rightarrow 'b$) $\Rightarrow$ $'d \Rightarrow 'd$ **and** *to-set'* :: $'d \Rightarrow 'b$ *set* +
  fixes
    map-to :: ($'a \Rightarrow 'b$) $\Rightarrow$ $'c \Rightarrow 'd$ **and**
    map-from :: ($'b \Rightarrow 'a$) $\Rightarrow$ $'d \Rightarrow 'c$
  assumes
    to-set-map-from [*simp*]: $\bigwedge f\ d.$ *to-set* (*map-from f d*) = *f* ‘ *to-set' d* **and**
    to-set-map-to [*simp*]: $\bigwedge f\ c.$ *to-set'* (*map-to f c*) = *f* ‘ *to-set c* **and**
    conversion-map-comp [*simp*]: $\bigwedge c\ f\ g.$ *map-from f* (*map-to g c*) = *map* ($\lambda x.$ *f* (*g x*)) *c* **and**
    conversion-map-comp' [*simp*]: $\bigwedge d\ f\ g.$ *map-to f* (*map-from g d*) = *map'* ($\lambda x.$ *f*

$(g\ x))\ d$

end
theory *Natural-Magma-Functor*
  imports *Natural-Magma Natural-Functor*
begin

locale *natural-magma-functor* = *natural-magma* + *natural-functor* +
  assumes
    map-wrap: $\bigwedge f\ a.\ map\ f\ (wrap\ a) = wrap\ (f\ a)$ **and**
    map-plus: $\bigwedge f\ b\ b'.\ map\ f\ (plus\ b\ b') = plus\ (map\ f\ b)\ (map\ f\ b')$
begin

lemma *map-add*: $\bigwedge f\ a\ b.\ map\ f\ (add\ a\ b) = add\ (f\ a)\ (map\ f\ b)$
  unfolding *add-def*
  using *map-plus map-wrap*
  by *simp*

end

end
theory *Functional-Substitution-Lifting*
  imports *Functional-Substitution Natural-Magma-Functor*
begin

locale *functional-substitution-lifting* =
  sub: *functional-substitution* **where** *subst* = *sub-subst* **and** *vars* = *sub-vars* +
  natural-functor **where** *map* = *map* **and** *to-set* = *to-set*
  for
    sub-vars :: $'sub \Rightarrow 'var\ set$ **and**
    sub-subst :: $'sub \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'sub$ **and**
    map :: $('sub \Rightarrow 'sub) \Rightarrow 'expr \Rightarrow 'expr$ **and**
    to-set :: $'expr \Rightarrow 'sub\ set$
begin

definition *vars* :: $'expr \Rightarrow 'var\ set$ **where**
  $vars\ expr \equiv \bigcup\ (sub\text{-}vars\ `\ to\text{-}set\ expr)$

notation *sub-subst* (**infixl** $\cdot_s$ *70*)

definition *subst* :: $'expr \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'expr$ (**infixl** $\cdot$ *70*) **where**
  $expr \cdot \sigma \equiv map\ (\lambda sub.\ sub \cdot_s \sigma)\ expr$

lemma *map-id-cong* [*simp*]:
  assumes $\bigwedge sub.\ sub \in to\text{-}set\ expr \Longrightarrow f\ sub = sub$
  shows $map\ f\ expr = expr$
  using *assms*
  by *simp*

```
lemma to-set-map-not-ident:
  assumes sub ∈ to-set expr f sub ∉ to-set expr
  shows map f expr ≠ expr
  using assms
  by (metis rev-image-eqI to-set-map)

lemma subst-in-to-set-subst [intro]:
  assumes sub ∈ to-set expr
  shows sub ·s σ ∈ to-set (expr · σ)
  unfolding subst-def to-set-map
  using assms
  by simp

sublocale functional-substitution where subst = (·) and vars = vars
proof unfold-locales
  fix expr σ1 σ2

  show expr · (σ1 ⊙ σ2) = expr · σ1 · σ2
    unfolding subst-def map-comp comp-apply sub.subst-comp-subst
    ..
next
  fix expr
  show expr · id-subst = expr
    using map-ident
    unfolding subst-def sub.subst-id-subst.
next
  fix expr
  assume vars expr = {}
  then show ∀ σ. expr · σ = expr
    unfolding vars-def subst-def
    by simp
next
  fix expr and σ1 σ2 :: 'var ⇒ 'base
  assume ⋀var. var ∈ vars expr ⟹ σ1 var = σ2 var

  then show expr · σ1 = expr · σ2
    unfolding vars-def subst-def
    using map-cong sub.subst-eq
    by (meson UN-I)
qed

lemma ground-subst-iff-sub-ground-subst [simp]: is-ground-subst γ ⟷ sub.is-ground-subst
γ
proof(unfold is-ground-subst-def sub.is-ground-subst-def, intro iffI allI)
  fix sub
  assume all-ground: ∀ expr. is-ground (expr · γ)
  show sub.is-ground (sub ·s γ)
  proof(rule ccontr)
    assume sub-not-ground: ¬sub.is-ground (sub ·s γ)
```

```
    then obtain expr where sub ∈ to-set expr
      using exists-functor by blast

    then have ¬is-ground (expr · γ)
      using sub-not-ground to-set-map
      unfolding subst-def vars-def
      by auto

    then show False
      using all-ground
      by blast
  qed
next
  fix expr
  assume ∀ sub. sub.is-ground (sub ·s γ)

  then show is-ground (expr · γ)
    unfolding vars-def subst-def
    using to-set-map
    by simp
qed

lemma to-set-is-ground [intro]:
  assumes sub ∈ to-set expr is-ground expr
  shows sub.is-ground sub
  using assms
  by (simp add: vars-def)

lemma to-set-is-ground-subst:
  assumes sub ∈ to-set expr is-ground (expr · γ)
  shows sub.is-ground (sub ·s γ)
  using assms
  by (meson subst-in-to-set-subst to-set-is-ground)

lemma subst-empty:
  assumes to-set expr' = {}
  shows expr · σ = expr' ⟷ expr = expr'
  using assms map-id-cong to-set-map
  unfolding subst-def
  by (metis empty-iff image-is-empty)

lemma empty-is-ground:
  assumes to-set expr = {}
  shows is-ground expr
  using assms
  by (simp add: vars-def)

lemma to-set-image: to-set (expr · σ) = (λa. a ·s σ) ` to-set expr
```

unfolding *subst-def to-set-map*..

lemma *to-set-subset-vars-subset*:
assumes *to-set expr* ⊆ *to-set expr'*
shows *vars expr* ⊆ *vars expr'*
using *assms*
unfolding *vars-def*
by *blast*

lemma *to-set-subset-is-ground*:
assumes *to-set expr'* ⊆ *to-set expr is-ground expr*
shows *is-ground expr'*
using *assms to-set-subset-vars-subset* **by** *blast*

end

locale *based-functional-substitution-lifting* =
functional-substitution-lifting +
base: *base-functional-substitution* **where** *subst* = *base-subst* **and** *vars* = *base-vars* +
sub: *based-functional-substitution* **where** *subst* = *sub-subst* **and** *vars* = *sub-vars*
begin

sublocale *based-functional-substitution* **where** *subst* = *subst* **and** *vars* = *vars*
proof *unfold-locales*
fix γ

show *is-ground-subst* γ = *base.is-ground-subst* γ
using *ground-subst-iff-sub-ground-subst sub.ground-subst-iff-base-ground-subst*
by *blast*
next
fix *expr* ϱ

show *vars* (*expr* · ϱ) = $\bigcup$ (*base-vars* ‘ ϱ ‘ *vars expr*)
using *sub.vars-subst*
unfolding *subst-def vars-def*
by *simp*
qed

end

locale *finite-variables-lifting* =
sub: *finite-variables* **where** *vars* = *sub-vars* :: *'sub* ⇒ *'var set* **and** *subst* = *sub-subst* +
finite-natural-functor **where** *to-set* = *to-set* :: *'expr* ⇒ *'sub set* +
functional-substitution-lifting
begin

abbreviation *to-fset* :: *'expr* ⇒ *'sub fset* **where**

to-fset expr ≡ *Abs-fset* (*to-set expr*)

sublocale *finite-variables* **where** *vars* = *vars* **and** *subst* = *subst*
by *unfold-locales* (*auto simp*: *vars-def finite-to-set*)

lemma *fset-to-fset* [*simp*]: *fset* (*to-fset expr*) = *to-set expr*
using *Abs-fset-inverse finite-to-set*
by *blast*

lemma *to-fset-map*: *to-fset* (*map f expr*) = *f* |'| *to-fset expr*
using *to-set-map*
by (*metis fset.set-map fset-inverse fset-to-fset*)

lemma *to-fset-is-ground-subst*:
assumes *sub* |∈| *to-fset expr is-ground* (*subst expr* γ)
shows *sub.is-ground* (*sub* $\cdot_s$ γ)
using *assms*
by (*simp add*: *to-set-is-ground-subst*)

end

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

sublocale *renaming-variables* **where** *subst* = *subst* **and** *vars* = *vars*
proof *unfold-locales*
fix *expr* ϱ
assume *sub.is-renaming* ϱ

then show *vars* (*expr* · ϱ) = *rename* ϱ ' *vars expr*
using *sub.rename-variables*
unfolding *vars-def subst-def to-set-map*
by *fastforce*
qed (*rule sub.is-renaming-iff*)

end

locale *based-renaming-variables-lifting* =
renaming-variables-lifting +
based-functional-substitution-lifting +
base: *renaming-variables* **where** *vars* = *base-vars* **and** *subst* = *base-subst*

locale *variables-in-base-imgu-lifting* =
based-functional-substitution-lifting +
sub: *variables-in-base-imgu* **where** *vars* = *sub-vars* **and** *subst* = *sub-subst*
begin

sublocale *variables-in-base-imgu* **where** *subst* = *subst* **and** *vars* = *vars*
proof *unfold-locales*
  fix *expr* μ *unifications*
  assume *imgu*:
    base.is-imgu μ *unifications*
    finite unifications
    $\forall$ *unification* $\in$ *unifications*. *finite unification*

  show *vars* (*expr* $\cdot$ μ) $\subseteq$ *vars expr* $\cup$ $\bigcup$ (*base-vars* ' $\bigcup$ *unifications*)
    using *sub.variables-in-base-imgu*[*OF imgu*]
    unfolding *vars-def subst-def to-set-map*
    by *auto*
qed

end

locale *grounding-lifting* =
 functional-substitution-lifting **where** *sub-vars* = *sub-vars* **and** *sub-subst* = *sub-subst*
and
  map = *map* +
  sub: *grounding* **where** *vars* = *sub-vars* **and** *subst* = *sub-subst* **and** *to-ground* = *sub-to-ground* **and**
  from-ground = *sub-from-ground* +
 natural-functor-conversion **where** *map* = *map* **and** *map-to* = *to-ground-map* **and**
 map-from = *from-ground-map* **and** *map'* = *ground-map* **and** *to-set'* = *to-set-ground*
  for
    sub-to-ground :: $'sub \Rightarrow 'sub_G$ **and**
    sub-from-ground :: $'sub_G \Rightarrow 'sub$ **and**
    sub-vars :: $'sub \Rightarrow 'var\ set$ **and**
    sub-subst :: $'sub \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'sub$ **and**
    map :: $('sub \Rightarrow 'sub) \Rightarrow 'expr \Rightarrow 'expr$ **and**
    to-ground-map :: $('sub \Rightarrow 'sub_G) \Rightarrow 'expr \Rightarrow 'expr_G$ **and**
    from-ground-map :: $('sub_G \Rightarrow 'sub) \Rightarrow 'expr_G \Rightarrow 'expr$ **and**
    ground-map :: $('sub_G \Rightarrow 'sub_G) \Rightarrow 'expr_G \Rightarrow 'expr_G$ **and**
    to-set-ground :: $'expr_G \Rightarrow 'sub_G\ set$
begin

definition *to-ground* :: $'expr \Rightarrow 'expr_G$ **where**
  to-ground expr $\equiv$ *to-ground-map sub-to-ground expr*

definition *from-ground* :: $'expr_G \Rightarrow 'expr$ **where**
  from-ground $expr_G$ $\equiv$ *from-ground-map sub-from-ground* $expr_G$

sublocale *grounding*
 where *vars* = *vars* **and** *subst* = *subst* **and** *to-ground* = *to-ground* **and** *from-ground* = *from-ground*
proof *unfold-locales*

  {

```
    fix expr

    assume is-ground expr

    then have ∀ sub∈to-set expr. sub ∈ range sub-from-ground
      by (simp add: sub.is-ground-iff-range-from-ground vars-def)

    then have ∀ sub∈to-set expr. ∃ sub_G. sub-from-ground sub_G = sub
      by fast

    then have ∃ expr_G. from-ground expr_G = expr
      unfolding from-ground-def
      by (metis conversion-map-comp map-id-cong)

    then have expr ∈ range from-ground
      unfolding from-ground-def
      by blast
  }

  moreover {
    fix expr var

    assume var ∈ vars (from-ground expr)

    then have False
      unfolding vars-def from-ground-def
      using sub.ground-is-ground to-set-map-from by auto
  }

  ultimately show {expr. is-ground expr} = range from-ground
    by blast
next
  fix expr_G
  show to-ground (from-ground expr_G) = expr_G
    unfolding from-ground-def to-ground-def
    by simp
qed

lemma to-set-from-ground: to-set (from-ground expr) = sub-from-ground ‘ (to-set-ground
expr)
  unfolding from-ground-def
  by simp

lemma sub-in-ground-is-ground:
  assumes sub ∈ to-set (from-ground expr)
  shows sub.is-ground sub
  using assms
  by (simp add: to-set-is-ground)
```

lemma *ground-sub-in-ground*:
  $sub \in$ *to-set-ground expr* $\longleftrightarrow$ *sub-from-ground sub* $\in$ *to-set* (*from-ground expr*)
  by (*simp add*: *inj-image-mem-iff sub.inj-from-ground to-set-from-ground*)

lemma *ground-sub*:
  ($\forall$ *sub* $\in$ *to-set* (*from-ground* $expr_G$). *P sub*) $\longleftrightarrow$
   ($\forall$ sub_G $\in$ *to-set-ground* $expr_G$. *P* (*sub-from-ground* sub_G))
  by (*simp add*: *to-set-from-ground*)

end

locale *all-subst-ident-iff-ground-lifting* =
  finite-variables-lifting **where** *map* = *map* +
  sub: *all-subst-ident-iff-ground* **where** *subst* = *sub-subst* **and** *vars* = *sub-vars*
for *map* :: ('*sub* ⇒ '*sub*) ⇒ '*expr* ⇒ '*expr*
begin

sublocale *all-subst-ident-iff-ground* **where** *subst* = *subst* **and** *vars* = *vars*
proof *unfold-locales*
  fix *expr*

  show *is-ground expr* $\longleftrightarrow$ ($\forall \sigma$. *expr* $\cdot\ \sigma$ = *expr*)
  proof(*rule iffI allI*)
    assume *is-ground expr*
    then show $\forall \sigma$. *expr* $\cdot\ \sigma$ = *expr*
      by *simp*
  next
    assume *all-subst-ident*: $\forall \sigma$. *expr* $\cdot\ \sigma$ = *expr*

    show *is-ground expr*
    proof(*rule ccontr*)
      assume ¬*is-ground expr*

      then obtain *sub* **where** *sub*: *sub* $\in$ *to-set expr* ¬*sub.is-ground sub*
        unfolding *vars-def*
        by *blast*

      then obtain σ **where** *sub* $\cdot_s\ \sigma \neq$ *sub* **and** *sub* $\cdot_s\ \sigma \notin$ *to-set expr*
        using *sub.exists-non-ident-subst finite-to-set*
        by *blast*

      then show *False*
        using *sub subst-in-to-set-subst all-subst-ident*
        by *metis*
    qed
  qed
next
  fix *expr* :: '*expr* **and** *S* :: '*expr set*

assume *finite*: *finite S* **and** *not-ground*: ¬*is-ground expr*

then have *finite-subs*: *finite* (⋃(*to-set* ‘ *insert expr S*))
using *finite-to-set* **by** *blast*

obtain *sub* **where** *sub*: *sub* ∈ *to-set expr* **and** *sub-not-ground*: ¬*sub.is-ground sub*
using *not-ground*
unfolding *vars-def*
by *blast*

obtain σ **where** σ*-not-ident*: *sub* $\cdot_s$ σ ≠ *sub sub* $\cdot_s$ σ ∉ ⋃ (*to-set* ‘ *insert expr S*)
using *sub.exists-non-ident-subst*[*OF finite-subs sub-not-ground*]
by *blast*

then have *expr* · σ ≠ *expr*
using *sub*
unfolding *subst-def*
by (*simp add*: *to-set-map-not-ident*)

moreover have *expr* · σ ∉ *S*
using σ*-not-ident*(*2*) *sub to-set-map*
unfolding *subst-def*
by *auto*

ultimately show ∃σ. *expr* · σ ≠ *expr* ∧ *expr* · σ ∉ *S*
by *blast*
qed

end

locale *natural-magma-functional-substitution-lifting* =
functional-substitution-lifting + *natural-magma*
begin

lemma *vars-add* [*simp*]:
vars (*add sub expr*) = *sub-vars sub* ∪ *vars expr*
unfolding *vars-def*
using *to-set-add* **by** *auto*

lemma *vars-plus* [*simp*]:
vars (*plus expr expr′*) = *vars expr* ∪ *vars expr′*
unfolding *vars-def*
by *simp*

lemma *is-ground-add* [*simp*]:
is-ground (*add sub expr*) ⟷ *sub.is-ground sub* ∧ *is-ground expr*
by *simp*

end

locale *natural-magma-functor-functional-substitution-lifting* =
natural-magma-functional-substitution-lifting + *natural-magma-functor*
begin

lemma *add-subst* [*simp*]:
$(add\ sub\ expr) \cdot \sigma = add\ (sub \cdot_s \sigma)\ (expr \cdot \sigma)$
unfolding *subst-def*
using *map-add*
by *simp*

lemma *plus-subst* [*simp*]: $(plus\ expr\ expr') \cdot \sigma = plus\ (expr \cdot \sigma)\ (expr' \cdot \sigma)$
unfolding *subst-def*
using *map-plus*
by *simp*

end

locale *natural-magma-grounding-lifting* =
grounding-lifting +
natural-magma-functional-substitution-lifting +
ground: *natural-magma* **where**
to-set = *to-set-ground* **and** *plus* = *plus-ground* **and** *wrap* = *wrap-ground* **and**
add = *add-ground*
for *plus-ground wrap-ground add-ground* +
assumes
to-ground-plus [*simp*]:
$\bigwedge$*expr expr'*. *to-ground* (*plus expr expr'*) = *plus-ground* (*to-ground expr*) (*to-ground*
expr') **and**
wrap-from-ground: $\bigwedge$*sub*. *from-ground* (*wrap-ground sub*) = *wrap* (*sub-from-ground*
sub) **and**
wrap-to-ground: $\bigwedge$*sub*. *to-ground* (*wrap sub*) = *wrap-ground* (*sub-to-ground sub*)
begin

lemma *from-ground-plus* [*simp*]:
from-ground (*plus-ground expr expr'*) = *plus* (*from-ground expr*) (*from-ground*
expr')
using *to-ground-plus*
by (*metis Un-empty-left from-ground-inverse ground-is-ground to-ground-inverse*
vars-plus)

lemma *from-ground-add* [*simp*]:
from-ground (*add-ground sub expr*) = *add* (*sub-from-ground sub*) (*from-ground*
expr)
unfolding *ground.add-def add-def*
using *from-ground-plus*
by (*simp add*: *wrap-from-ground*)

lemma *to-ground-add* [*simp*]:
to-ground (*add sub expr*) = *add-ground* (*sub-to-ground sub*) (*to-ground expr*)
unfolding *ground.add-def add-def*
using *from-ground-add wrap-to-ground*
by *simp*

lemma *ground-add*:
assumes *from-ground expr* = *add sub expr′*
shows *expr* = *add-ground* (*sub-to-ground sub*) (*to-ground expr′*)
using *assms*
by (*metis from-ground-inverse to-ground-add*)

end

locale *natural-magma-with-empty-grounding-lifting* =
natural-magma-grounding-lifting +
natural-magma-with-empty +
ground: *natural-magma-with-empty* **where**
to-set = *to-set-ground* **and** *plus* = *plus-ground* **and** *wrap* = *wrap-ground* **and**
add = *add-ground* **and**
empty = *empty-ground*
for *empty-ground* +
assumes *to-ground-empty* [*simp*]: *to-ground empty* = *empty-ground*
begin

lemmas *empty-magma-is-ground* [*simp*] = *empty-is-ground*[*OF to-set-empty*]

lemmas *magma-subst-empty* [*simp*] =
subst-ident-if-ground[*OF empty-magma-is-ground*]
subst-empty[*OF to-set-empty*]

lemma *from-ground-empty* [*simp*]: *from-ground empty-ground* = *empty*
using *to-ground-inverse*[*OF empty-magma-is-ground*]
by *simp*

lemma *to-ground-empty′* [*simp*]: *to-ground expr* = *empty-ground* ⟷ *expr* = *empty*
using *from-ground-empty to-ground-empty ground.to-set-empty to-ground-inverse*
unfolding *to-ground-def vars-def*
by *fastforce*

lemma *from-ground-empty′* [*simp*]: *from-ground expr* = *empty* ⟷ *expr* = *empty-ground*
using *from-ground-empty from-ground-eq*
unfolding *from-ground-def*
by *auto*

end

end
theory *Functional-Substitution-Lifting-Example*
imports
Functional-Substitution-Lifting
Functional-Substitution-Example
begin

Lifting of properties from term to equations (modelled as pairs)

type-synonym $('f,'v)$ *equation* = $('f, 'v)$ *term* × $('f, 'v)$ *term*

All property locales have corresponding lifting locales

locale *lifting-term-subst-properties* =
based-functional-substitution-lifting **where**
id-subst = *Var* **and** *comp-subst* = *subst-compose* **and** *base-subst* = *subst-apply-term* **and**
base-vars = *vars-term* :: $('f, 'v)$ *term* ⇒ *'v set* +
finite-variables-lifting **where** *id-subst* = *Var* **and** *comp-subst* = *subst-compose*

global-interpretation *equation-subst*:
lifting-term-subst-properties **where**
sub-vars = *vars-term* **and** *sub-subst* = *subst-apply-term* **and** *map* = λf. *map-prod* *f f* **and**
to-set = *set-prod*
by *unfold-locales fastforce+*

Lifted lemmas and defintions

thm
equation-subst.subst-reduntant-upd
equation-subst.subst-reduntant-if
equation-subst.vars-subst-subset

equation-subst.vars-def
equation-subst.subst-def

We can lift multiple levels

global-interpretation *equation-set-subst*:
lifting-term-subst-properties **where**
sub-vars = *equation-subst.vars* **and** *sub-subst* = *equation-subst.subst* **and** *map* = *fimage* **and**
to-set = *fset*
by *unfold-locales* (*auto simp*: *rev-image-eqI*)

Lifted lemmas and defintions

thm
equation-set-subst.subst-reduntant-upd
equation-set-subst.subst-reduntant-if
equation-set-subst.vars-subst-subset

equation-set-subst.vars-def
equation-set-subst.subst-def

end