

Abstract Substitutions as Monoid Actions

Martin Desharnais

March 16, 2025

Abstract

This entry provides a small, reusable, theory that specifies the abstract concept of substitution 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

1	General Results on Groups	1
2	Monoid	2
3	Semigroup Action	2
4	Monoid Action	3
5	Group Action	4
6	Assumption-free Substitution	4
7	Basic Substitution	7
7.1	Substitution Composition	8
7.2	Substitution Identity	8
7.3	Generalization	9
7.4	Substituting on Ground Expressions	9
7.5	Instances of Ground Expressions	10
7.6	Unifier of Ground Expressions	10
7.7	Ground Substitutions	11
7.8	IMGU is Idempotent and an MGU	11
7.9	IMGU can be used before unification	11
7.10	Groundings Idempotence	11
7.11	Instances of Substitution	11
7.12	Instances of Renamed Expressions	12

```

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  $\bigwedge a. inv\ (inv\ a) = a$ 
   $\langle proof \rangle$ 

```

```

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

```

```

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

```

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  $a \implies left\text{-inverse}\ a = (SOME\ a\text{-inv}. a\text{-inv} * a = \mathbf{1})$ 

```

```

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

```

```

lemma (in monoid) comp-left-inverse [simp]:
  is-left-invertible  $a \implies left\text{-inverse}\ a * a = \mathbf{1}$ 
   $\langle proof \rangle$ 

```

```

lemma (in monoid) comp-right-inverse [simp]:
  is-right-invertible  $a \implies a * right\text{-inverse}\ a = \mathbf{1}$ 
   $\langle proof \rangle$ 

```

```

lemma (in monoid) neutral-is-left-invertible [simp]:

```

is-left-invertible **1**
 ⟨*proof*⟩

lemma (in *monoid*) *neutral-is-right-invertible* [*simp*]:
is-right-invertible **1**
 ⟨*proof*⟩

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 ⇒ 'b ⇒ '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 ⇒ 'a ⇒ '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* \subseteq *right*: *right-semigroup-action* **where**
f = $\lambda x y. f y x$ **and** *action* = $\lambda x y. action y x$
 ⟨*proof*⟩

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

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

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

4 Monoid Action

locale *left-monoid-action* = *monoid* +
fixes *action* :: 'a ⇒ 'b ⇒ '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 ⇒ 'a ⇒ '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* \subseteq *left-semigroup-action*
<proof>

sublocale *right-monoid-action* \subseteq *right-semigroup-action*
<proof>

sublocale *left-monoid-action* \subseteq *right: right-monoid-action* **where**
f = $\lambda x y. f y x$ **and** *action* = $\lambda x y. action y x$
<proof>

sublocale *right-monoid-action* \subseteq *left: left-monoid-action* **where**
f = $\lambda x y. f y x$ **and** *action* = $\lambda x y. action y x$
<proof>

lemma (**in** *right-monoid-action*) *lifting-monoid-action-to-set*:
right-monoid-action (*) **1** ($\lambda X a. (\lambda x. action x a) \text{ ' } X$)
<proof>

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>

5 Group Action

locale *left-group-action* = *group* +
fixes *action* :: 'a ⇒ 'b ⇒ '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 ⇒ 'a ⇒ '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* \subseteq *left-monoid-action*
<proof>

sublocale *right-group-action* \subseteq *right-monoid-action*
<proof>

sublocale *left-group-action* \subseteq *right: right-group-action* **where**

$f = \lambda x y. f y x$ **and** $action = \lambda x y. action y x$
 ⟨proof⟩

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

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

abbreviation $set\text{-prod}$ **where**
 $set\text{-prod} \equiv \lambda(t, t'). \{t, t'\}$

6 Assumption-free Substitution

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

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

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

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

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

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

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

definition *instances* :: 'x ⇒ 'x set **where**
instances x = {y. generalizes x y}

definition *instances-set* :: 'x set ⇒ 'x set **where**
instances-set X = (⋃ x ∈ X. instances x)

definition *ground-instances* :: 'x ⇒ 'x set **where**
ground-instances x = {x_G ∈ instances x. is-ground x_G}

definition *ground-instances-set* :: 'x set ⇒ 'x set **where**
ground-instances-set X = {x_G ∈ instances-set X. is-ground x_G}

lemma *ground-instances-set-eq-Union-ground-instances*:
ground-instances-set X = (⋃ x ∈ X. *ground-instances* x)
 ⟨proof⟩

lemma *ground-instances-eq-Collect-subst-grounding*:
ground-instances x = {x · γ | γ. is-ground (x · γ)}
 ⟨proof⟩

lemma *mem-ground-instancesE[elim]*:
fixes x x_G :: 'x
assumes x_G ∈ *ground-instances* x
obtains γ :: 's **where** x_G = x · γ **and** is-ground (x · γ)
 ⟨proof⟩

lemma *mem-ground-instances-setE[elim]*:
fixes x_G :: 'x **and** X :: 'x set
assumes x_G ∈ *ground-instances-set* X
obtains x :: 'x **and** γ :: 's **where** x ∈ X **and** x_G = x · γ **and** is-ground (x · γ)
 ⟨proof⟩

definition *is-unifier* :: 's ⇒ 'x set ⇒ bool **where**
is-unifier v X ⟷ card (subst-set X v) ≤ 1

definition *is-unifier-set* :: 's ⇒ 'x set set ⇒ bool **where**
is-unifier-set v XX ⟷ (∀ X ∈ XX. is-unifier v X)

definition *is-mgu* :: 's ⇒ 'x set set ⇒ bool **where**
is-mgu μ XX ⟷ is-unifier-set μ XX ∧ (∀ v. is-unifier-set v XX ⟶ (∃ σ. μ ⊙ σ = v))

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

lemma *is-unifier-iff-if-finite*:
assumes finite X
shows is-unifier σ X ⟷ (∀ x ∈ X. ∀ y ∈ X. x · σ = y · σ)

<proof>

lemma *is-unifier-singleton[simp]*: *is-unifier* v $\{x\}$
<proof>

lemma *is-unifier-set-empty[simp]*:
is-unifier-set σ $\{\}$
<proof>

lemma *is-unifier-set-insert*:
is-unifier-set σ (*insert* X XX) \longleftrightarrow *is-unifier* σ X \wedge *is-unifier-set* σ XX
<proof>

lemma *is-unifier-set-insert-singleton[simp]*:
is-unifier-set σ (*insert* $\{x\}$ XX) \longleftrightarrow *is-unifier-set* σ XX
<proof>

lemma *is-mgu-insert-singleton[simp]*: *is-mgu* μ (*insert* $\{x\}$ XX) \longleftrightarrow *is-mgu* μ XX
<proof>

lemma *is-imgu-insert-singleton[simp]*: *is-imgu* μ (*insert* $\{x\}$ XX) \longleftrightarrow *is-imgu* μ XX
<proof>

lemma *subst-set-empty[simp]*: *subst-set* $\{\}$ $\sigma = \{\}$
<proof>

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

lemma *subst-set-union[simp]*: *subst-set* ($X1 \cup X2$) $\sigma =$ *subst-set* $X1$ $\sigma \cup$ *subst-set* $X2$ σ
<proof>

lemma *subst-list-Nil[simp]*: *subst-list* \square $\sigma = \square$
<proof>

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

lemma *subst-list-append[simp]*: *subst-list* ($xs_1 @ xs_2$) $\sigma =$ *subst-list* xs_1 $\sigma @$ *subst-list* xs_2 σ
<proof>

lemma *is-unifier-set-union*:
is-unifier-set v ($XX_1 \cup XX_2$) \longleftrightarrow *is-unifier-set* v XX_1 \wedge *is-unifier-set* v XX_2
<proof>

lemma *is-unifier-subset*: *is-unifier* v $A \implies$ *finite* $A \implies B \subseteq A \implies$ *is-unifier* v B

<proof>

lemma *is-ground-set-subset*: *is-ground-set* $A \implies B \subseteq A \implies$ *is-ground-set* B

<proof>

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

<proof>

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

<proof>

end

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 $a \iff$ *comp-subst* a $a = a$

lemma *is-idem-id-subst [simp]*: *is-idem* *id-subst*

<proof>

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 ⇒ 's ⇒ 's (**infixl** <⊙> 70) **and**
id-subst :: 's **and**
subst :: 'x ⇒ 's ⇒ 'x (**infixl** <·> 69) **and**

— Predicate identifying the fixed elements w.r.t. the monoid action
is-ground :: 'x ⇒ bool +

assumes
all-subst-ident-if-ground: *is-ground* x ⇒ (∀ σ. x · σ = x)

begin

sublocale *comp-subst-set*: *right-monoid-action comp-subst id-subst subst-set*
<proof>

sublocale *comp-subst-list*: *right-monoid-action comp-subst id-subst subst-list*
<proof>

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* {}
<proof>

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

lemma *is-mgu-id-subst-empty[simp]*: *is-mgu id-subst* {}
<proof>

lemma *is-imgu-id-subst-empty[simp]*: *is-imgu id-subst* {}
<proof>

lemma *is-unifier-id-subst*: *is-unifier id-subst* X ⇔ *card* X ≤ 1
<proof>

lemma *is-unifier-set-id-subst*: *is-unifier-set id-subst* XX ⇔ (∀ X ∈ XX. *card* X ≤ 1)
<proof>

lemma *is-mgu-id-subst*: *is-mgu id-subst* XX ⇔ (∀ X ∈ XX. *card* X ≤ 1)
<proof>

lemma *is-imgu-id-subst*: *is-imgu id-subst* $XX \longleftrightarrow (\forall X \in XX. \text{card } X \leq 1)$
 ⟨*proof*⟩

7.3 Generalization

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

lemma *generalizes-antisym-if*:
assumes $\bigwedge \sigma_1 \sigma_2 x. x \cdot (\sigma_1 \odot \sigma_2) = x \implies x \cdot \sigma_1 = x$
shows $\bigwedge x y. \text{generalizes } x y \implies \text{generalizes } y x \implies x = y$
 ⟨*proof*⟩

lemma *order-generalizes-if*:
assumes $\bigwedge \sigma_1 \sigma_2 x. x \cdot (\sigma_1 \odot \sigma_2) = x \implies x \cdot \sigma_1 = x$
shows *class.order generalizes strictly-generalizes*
 ⟨*proof*⟩

7.4 Substituting on Ground Expressions

lemma *subst-ident-if-ground[simp]*: *is-ground* $x \implies x \cdot \sigma = x$
 ⟨*proof*⟩

lemma *subst-set-ident-if-ground[simp]*: *is-ground-set* $X \implies \text{subst-set } X \sigma = X$
 ⟨*proof*⟩

7.5 Instances of Ground Expressions

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

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

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

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

7.6 Unifier of Ground Expressions

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

lemma *ball-eq-constant-if-unifier*:

assumes *finite X and $x \in X$ and is-unifier $v X$ and is-ground-set X*
shows $\forall y \in X. y = x$
<proof>

lemma *is-mgu-unifies*:

assumes *is-mgu $\mu XX \forall X \in XX. finite X$*
shows $\forall X \in XX. \forall t \in X. \forall t' \in X. t \cdot \mu = t' \cdot \mu$
<proof>

corollary *is-mgu-unifies-pair*:

assumes *is-mgu $\mu \{\{t, t'\}\}$*
shows $t \cdot \mu = t' \cdot \mu$
<proof>

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

lemma *is-imgu-unifies*:

assumes *is-imgu $\mu XX \forall X \in XX. finite X$*
shows $\forall X \in XX. \forall t \in X. \forall t' \in X. t \cdot \mu = t' \cdot \mu$
<proof>

corollary *is-imgu-unifies-pair*:

assumes *is-imgu $\mu \{\{t, t'\}\}$*
shows $t \cdot \mu = t' \cdot \mu$
<proof>

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

7.7 Ground Substitutions

lemma *is-ground-subst-comp-left*: *is-ground-subst $\sigma \implies is-ground-subst (\sigma \odot \tau)$*
<proof>

lemma *is-ground-subst-comp-right*: *is-ground-subst $\tau \implies is-ground-subst (\sigma \odot \tau)$*
<proof>

lemma *is-ground-subst-is-ground*:

assumes *is-ground-subst γ*
shows *is-ground $(t \cdot \gamma)$*
<proof>

7.8 IMGU is Idempotent and an MGU

lemma *is-imgu-iff-is-idem-and-is-mgu*: *is-imgu $\mu XX \iff is-idem \mu \wedge is-mgu \mu XX$*
<proof>

7.9 IMGU can be used before unification

lemma *subst-imgu-subst-unifier*:

assumes *unif*: *is-unifier* v X and *imgu*: *is-imgu* μ $\{X\}$ and $x \in X$

shows $x \cdot \mu \cdot v = x \cdot v$

<proof>

7.10 Groundings Idempotence

lemma *image-ground-instances-ground-instances*:

ground-instances ' *ground-instances* $x = (\lambda x. \{x\})$ ' *ground-instances* x

<proof>

lemma *grounding-of-set-grounding-of-set-idem[simp]*:

ground-instances-set (*ground-instances-set* X) = *ground-instances-set* X

<proof>

7.11 Instances of Substitution

lemma *instances-subst*:

instances $(x \cdot \sigma) \subseteq$ *instances* x

<proof>

lemma *instances-set-subst-set*:

instances-set (*subst-set* X σ) \subseteq *instances-set* X

<proof>

lemma *ground-instances-subst*:

ground-instances $(x \cdot \sigma) \subseteq$ *ground-instances* x

<proof>

lemma *ground-instances-set-subst-set*:

ground-instances-set (*subst-set* X σ) \subseteq *ground-instances-set* X

<proof>

7.12 Instances of Renamed Expressions

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

is-renaming $\varrho \implies$ *instances* $(x \cdot \varrho) =$ *instances* x

<proof>

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

is-renaming $\varrho \implies$ *instances-set* (*subst-set* X ϱ) = *instances-set* X

<proof>

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

is-renaming $\varrho \implies$ *ground-instances* $(x \cdot \varrho) =$ *ground-instances* x

<proof>

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

is-renaming $\varrho \implies \text{ground-instances-set } (\text{subst-set } X \ \varrho) = \text{ground-instances-set } X$
 ⟨proof⟩

end

end

theory *Substitution-First-Order-Term*

imports

Substitution

First-Order-Terms.Unification

begin

abbreviation *is-ground-trm* **where**

is-ground-trm $t \equiv \text{vars-term } t = \{\}$

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

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

global-interpretation *term-subst*: *substitution* **where**

subst = *subst-apply-term* **and** *id-subst* = *Var* **and** *comp-subst* = *subst-compose*

and

is-ground = *is-ground-trm*

⟨proof⟩

lemma *term-subst-is-unifier-iff-unifiers-Times*:

assumes *finite* X

shows *term-subst.is-unifier* $\mu \ X \longleftrightarrow \mu \in \text{unifiers } (X \times X)$
 ⟨proof⟩

lemma *term-subst-is-unifier-set-iff-unifiers-Union-Times*:

assumes $\forall X \in XX. \text{finite } X$

shows *term-subst.is-unifier-set* $\mu \ XX \longleftrightarrow \mu \in \text{unifiers } (\bigcup X \in XX. X \times X)$
 ⟨proof⟩

lemma *term-subst-is-mgu-iff-is-mgu-Union-Times*:

assumes *fin*: $\forall X \in XX. \text{finite } X$

shows *term-subst.is-mgu* $\mu \ XX \longleftrightarrow \text{is-mgu } \mu (\bigcup X \in XX. X \times X)$
 ⟨proof⟩

lemma *term-subst-is-imagu-iff-is-imagu-Union-Times*:

assumes $\forall X \in XX. \text{finite } X$

shows *term-subst.is-imagu* $\mu \ XX \longleftrightarrow \text{is-imagu } \mu (\bigcup X \in XX. X \times X)$
 ⟨proof⟩

lemma *range-vars-subset-if-is-imgu:*

assumes *term-subst.is-imgu* μ $XX \forall X \in XX. \text{finite } X \text{ finite } XX$

shows *range-vars* $\mu \subseteq (\bigcup t \in \bigcup XX. \text{vars-term } t)$

<proof>

lemma *term-subst-is-renaming-iff:*

term-subst.is-renaming $\varrho \longleftrightarrow \text{inj } \varrho \wedge (\forall x. \text{is-Var } (\varrho x))$

<proof>

lemma *term-subst-is-renaming-iff-ex-inj-fun-on-vars:*

term-subst.is-renaming $\varrho \longleftrightarrow (\exists f. \text{inj } f \wedge \varrho = \text{Var} \circ f)$

<proof>

lemma *ground-imgu-equals:*

assumes *is-ground-trm* t_1 **and** *is-ground-trm* t_2 **and** *term-subst.is-imgu* $\mu \{\{t_1, t_2\}\}$

shows $t_1 = t_2$

<proof>

lemma *is-unifier-the-mgu:*

assumes $t \cdot \text{the-mgu } t' = t' \cdot \text{the-mgu } t'$

shows *term-subst.is-unifier* (*the-mgu* t') $\{t, t'\}$

<proof>

lemma *obtains-imgu-from-unifier-and-the-mgu:*

fixes $v :: ('f, 'v) \text{subst}$

assumes $t \cdot v = t' \cdot v$ $P t t' (\text{Unification.the-mgu } t')$

obtains $\mu :: ('f, 'v) \text{subst}$

where $v = \mu \circ_s v$ *term-subst.is-imgu* $\mu \{\{t, t'\}\} P t t' \mu$

<proof>

lemma *obtains-imgu:*

fixes $v :: ('f, 'v) \text{subst}$

assumes $t \cdot v = t' \cdot v$

obtains $\mu :: ('f, 'v) \text{subst}$

where $v = \mu \circ_s v$ *term-subst.is-imgu* $\mu \{\{t, t'\}\}$

<proof>

lemma *is-renaming-if-term-subst-is-renaming:*

assumes *term-subst.is-renaming* ϱ

shows *Term.is-renaming* ϱ

<proof>

lemma *is-mgu-iff-term-subst-is-imgu-image-set-prod:*

fixes $\mu :: ('f, 'v) \text{subst}$ **and** $X :: (('f, 'v) \text{term} \times ('f, 'v) \text{term}) \text{set}$

shows *Unifiers.is-imgu* $\mu X \longleftrightarrow \text{term-subst.is-imgu } \mu (\text{set-prod } 'X)$

<proof>

```

lemma the-mgu-term-subst-is-imgu:
  fixes  $v :: ('f, 'v) \text{subst}$ 
  assumes  $s \cdot v = t \cdot v$ 
  shows  $\text{term-subst.is-imgu } (\text{Unification.the-mgu } s \ t) \ \{\{s, t\}\}$ 
   $\langle \text{proof} \rangle$ 

end
theory Substitution-HOL-ex-Unification
  imports
    Substitution
    HOL-ex.Unification
begin

no-notation Comb (infix  $\langle \cdot \rangle$  60)

quotient-type  $'a \ \text{subst} = ('a \times 'a \ \text{trm}) \ \text{list} / (\doteq)$ 
   $\langle \text{proof} \rangle$ 

lift-definition  $\text{subst-comp} :: 'a \ \text{subst} \Rightarrow 'a \ \text{subst} \Rightarrow 'a \ \text{subst}$  (infixl  $\langle \odot \rangle$  67)
  is Unification.comp
   $\langle \text{proof} \rangle$ 

definition  $\text{subst-id} :: 'a \ \text{subst}$  where
   $\text{subst-id} = \text{abs-subst } []$ 

global-interpretation  $\text{subst-comp}: \text{monoid } \text{subst-comp } \text{subst-id}$ 
   $\langle \text{proof} \rangle$ 

lift-definition  $\text{subst-apply} :: 'a \ \text{trm} \Rightarrow 'a \ \text{subst} \Rightarrow 'a \ \text{trm}$ 
  is Unification.subst
   $\langle \text{proof} \rangle$ 

abbreviation  $\text{is-ground-trm}$  where
   $\text{is-ground-trm } t \equiv \text{vars-of } t = \{\}$ 

global-interpretation  $\text{term-subst}: \text{substitution}$  where
   $\text{subst} = \text{subst-apply}$  and  $\text{id-subst} = \text{subst-id}$  and  $\text{comp-subst} = \text{subst-comp}$  and
   $\text{is-ground} = \text{is-ground-trm}$ 
   $\langle \text{proof} \rangle$ 

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

locale  $\text{functional-substitution} = \text{substitution}$  where
   $\text{subst} = \text{subst}$  and  $\text{is-ground} = \lambda \text{expr}. \text{vars } \text{expr} = \{\}$ 

```

for
 $subst :: 'expr \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'expr$ (**infixl** · 69) **and**
 $vars :: 'expr \Rightarrow 'var\ set +$
assumes
 $subst\text{-}eq: \bigwedge expr\ \sigma\ \tau. (\bigwedge x. x \in vars\ expr \implies \sigma\ x = \tau\ x) \implies expr \cdot \sigma = expr \cdot \tau$
begin

abbreviation *is-ground* **where** $is\text{-}ground\ expr \equiv vars\ expr = \{\}$

definition *vars-set* $:: 'expr\ set \Rightarrow 'var\ set$ **where**
 $vars\text{-}set\ exprs \equiv \bigcup expr \in exprs. vars\ expr$

lemma *subst-redundant-upd* [*simp*]:
assumes $var \notin vars\ expr$
shows $expr \cdot \sigma(var := update) = expr \cdot \sigma$
<proof>

lemma *subst-redundant-if* [*simp*]:
assumes $vars\ expr \subseteq vars'$
shows $expr \cdot (\lambda var. if\ var \in vars' then\ \sigma\ var\ else\ \sigma'\ var) = expr \cdot \sigma$
<proof>

lemma *subst-redundant-if'* [*simp*]:
assumes $vars\ expr \cap vars' = \{\}$
shows $expr \cdot (\lambda var. if\ var \in vars' then\ \sigma'\ var\ else\ \sigma\ var) = expr \cdot \sigma$
<proof>

lemma *subst-cannot-unground*:
assumes $\neg is\text{-}ground\ (expr \cdot \sigma)$
shows $\neg is\text{-}ground\ expr$
<proof>

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

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

lemma *subst-inv*:
assumes $\sigma \odot \sigma\text{-}inv = id\text{-}subst$
shows $expr \cdot \sigma \cdot \sigma\text{-}inv = expr$
<proof>

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

end

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

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

abbreviation *finite-vars* :: 'expr \Rightarrow 'var fset **where**
finite-vars expr $\equiv Abs_fset\ (vars\ expr)$

lemma *fset-finite-vars* [*simp*]: $fset\ (finite_vars\ expr) = vars\ expr$
<proof>

end

locale *renaming-variables* = *functional-substitution* +
assumes
is-renaming-iff: $\bigwedge \varrho. is_renaming\ \varrho \longleftrightarrow inj\ \varrho \wedge (\forall x. \exists x'. \varrho\ x = id_subst\ x')$
and
rename-variables: $\bigwedge expr\ \varrho. is_renaming\ \varrho \implies vars\ (expr \cdot \varrho) = rename\ \varrho\ ` (vars\ expr)$
begin

lemma *renaming-range-id-subst*:
assumes *is-renaming* ϱ
shows $\varrho\ x \in range\ id_subst$
<proof>

lemma *obtain-renamed-variable*:
assumes *is-renaming* ϱ
obtains x' **where** $\varrho\ x = id_subst\ x'$
<proof>

lemma *id-subst-rename* [*simp*]:
assumes *is-renaming* ϱ
shows $id_subst\ (rename\ \varrho\ x) = \varrho\ x$
<proof>

lemma *rename-variables-id-subst*:
assumes *is-renaming* ϱ
shows $id_subst\ ` vars\ (expr \cdot \varrho) = \varrho\ ` (vars\ expr)$
<proof>

lemma *surj-inv-renaming*:
assumes *is-renaming* ϱ
shows *surj* $(\lambda x. \text{inv } \varrho (\text{id-subst } x))$
 $\langle \text{proof} \rangle$

lemma *renaming-range*:
assumes *is-renaming* ϱ $x \in \text{vars } (\text{expr} \cdot \varrho)$
shows $\text{id-subst } x \in \text{range } \varrho$
 $\langle \text{proof} \rangle$

lemma *renaming-inv-into*:
assumes *is-renaming* ϱ $x \in \text{vars } (\text{expr} \cdot \varrho)$
shows $\varrho (\text{inv } \varrho (\text{id-subst } x)) = \text{id-subst } x$
 $\langle \text{proof} \rangle$

lemma *inv-renaming*:
assumes *is-renaming* ϱ
shows $\text{inv } \varrho (\varrho x) = x$
 $\langle \text{proof} \rangle$

lemma *renaming-inv-in-vars*:
assumes *is-renaming* ϱ $x \in \text{vars } (\text{expr} \cdot \varrho)$
shows $\text{inv } \varrho (\text{id-subst } x) \in \text{vars } \text{expr}$
 $\langle \text{proof} \rangle$

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: $\{\text{expr}. \text{is-ground } \text{expr}\} = \text{range } \text{from-ground}$

and

from-ground-inverse [*simp*]: $\bigwedge \text{expr}_G. \text{to-ground } (\text{from-ground } \text{expr}_G) = \text{expr}_G$

begin

definition *ground-instances'* :: 'expr \Rightarrow 'expr_G set **where**
ground-instances' *expr* = $\{\text{to-ground } (\text{expr} \cdot \gamma) \mid \gamma. \text{is-ground } (\text{expr} \cdot \gamma)\}$

lemma *ground-instances'-eq-ground-instances*:
ground-instances' *expr* = (*to-ground* ' *ground-instances* *expr*)
 $\langle \text{proof} \rangle$

lemma *to-ground-from-ground-id* [*simp*]: *to-ground* \circ *from-ground* = *id*
 $\langle \text{proof} \rangle$

lemma *surj-to-ground*: *surj* *to-ground*

<proof>

lemma *inj-from-ground: inj-on from-ground domain_G*
<proof>

lemma *inj-on-to-ground: inj-on to-ground (from-ground ‘ domain_G)*
<proof>

lemma *bij-betw-to-ground: bij-betw to-ground (from-ground ‘ domain_G) domain_G*
<proof>

lemma *bij-betw-from-ground: bij-betw from-ground domain_G (from-ground ‘ domain_G)*
<proof>

lemma *ground-is-ground [simp, intro]: is-ground (from-ground expr_G)*
<proof>

lemma *is-ground-iff-range-from-ground: is-ground expr \longleftrightarrow expr \in range from-ground*
<proof>

lemma *to-ground-inverse [simp]:*
assumes *is-ground expr*
shows *from-ground (to-ground expr) = expr*
<proof>

corollary *obtain-grounding:*
assumes *is-ground expr*
obtains *expr_G where from-ground expr_G = expr*
<proof>

lemma *from-ground-eq [simp]:*
from-ground expr = from-ground expr' \longleftrightarrow expr = expr'
<proof>

lemma *to-ground-eq [simp]:*
assumes *is-ground expr is-ground expr'*
shows *to-ground expr = to-ground expr' \longleftrightarrow expr = expr'*
<proof>

end

locale *base-functional-substitution = functional-substitution*
where *id-subst = id-subst and vars = vars*
for *id-subst :: 'var \Rightarrow 'expr and vars :: 'expr \Rightarrow 'var set +*
assumes
vars-subst-vars: \bigwedge expr ϱ . vars (expr \cdot ϱ) = \bigcup (vars ‘ ϱ ‘ vars expr) and
base-ground-exists: \exists expr. is-ground expr and
vars-id-subst: \bigwedge x. vars (id-subst x) = {x} and

comp-subst-iff: $\bigwedge \sigma \sigma' x. (\sigma \odot \sigma') x = \sigma x \cdot \sigma'$

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 \Rightarrow ('var \Rightarrow 'base) \Rightarrow 'base **and**
base-vars **and**
vars :: 'expr \Rightarrow 'var set +
assumes
ground-subst-iff-base-ground-subst [*simp*]: $\bigwedge \gamma. \text{is-ground-subst } \gamma \longleftrightarrow \text{base.is-ground-subst } \gamma$
and
vars-subst: $\bigwedge \text{expr } \varrho. \text{vars } (\text{expr} \cdot \varrho) = \bigcup (\text{base-vars } \text{' } \varrho \text{' } \text{vars expr})$
begin

lemma *is-grounding-iff-vars-grounded*:
is-ground (*expr* \cdot γ) \longleftrightarrow ($\forall \text{var} \in \text{vars expr}. \text{base.is-ground } (\gamma \text{ var})$)
⟨*proof*⟩

lemma *obtain-ground-subst*:
obtains γ
where *is-ground-subst* γ
⟨*proof*⟩

lemma *exists-ground-subst* [*intro*]: $\exists \gamma. \text{is-ground-subst } \gamma$
⟨*proof*⟩

lemma *ground-subst-extension*:
assumes *is-ground* (*expr* \cdot γ)
obtains γ'
where *expr* \cdot $\gamma = \text{expr} \cdot \gamma'$ **and** *is-ground-subst* γ'
⟨*proof*⟩

lemma *ground-subst-extension'*:
assumes *is-ground* (*expr* \cdot γ)
obtains γ'
where *expr* \cdot $\gamma = \text{expr} \cdot \gamma'$ **and** *base.is-ground-subst* γ'
⟨*proof*⟩

lemma *ground-subst-update* [*simp*]:
assumes *base.is-ground update is-ground* (*expr* \cdot γ)
shows *is-ground* (*expr* \cdot $\gamma(\text{var} := \text{update})$)
⟨*proof*⟩

lemma *ground-exists*: $\exists \text{expr}. \text{is-ground expr}$
⟨*proof*⟩

lemma *variable-grounding*:

assumes *is-ground* (*expr* · γ) *var* ∈ *vars expr*
shows *base.is-ground* (γ *var*)
 ⟨*proof*⟩

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

lemma *vars-subst-subset*: *vars* (*expr* · σ) ⊆ (*vars expr* − *subst-domain* σ) ∪
range-vars σ
 ⟨*proof*⟩

end

locale *variables-in-base-imgu* = *based-functional-substitution* +
assumes *variables-in-base-imgu*:
 ∧*expr* μ *unifications*.
 base.is-imgu μ *unifications* ⇒
 finite unifications ⇒
 ∀*unification* ∈ *unifications*. *finite unification* ⇒
 vars (*expr* · μ) ⊆ *vars expr* ∪ ($\bigcup (\text{base-vars } \text{'unifications})$)

context *base-functional-substitution*
begin

sublocale *based-functional-substitution*
where *base-subst* = *subst* **and** *base-vars* = *vars*
 ⟨*proof*⟩

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 $\wedge t. \text{term-subst.is-ground } t \longleftrightarrow \text{ground } t$

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

<proof>

Examples of generated lemmas and definitions

thm

term-subst.subst-redundant-upd
term-subst.subst-redundant-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 \Rightarrow 'a set **and**

plus :: 'b \Rightarrow 'b \Rightarrow 'b **and**

wrap :: 'a \Rightarrow 'b **and**

add

defines $\wedge a b. \text{add } a b \equiv \text{plus } (\text{wrap } a) b$

assumes

to-set-plus [*simp*]: $\wedge b b'. \text{to-set } (\text{plus } b b') = (\text{to-set } b) \cup (\text{to-set } b')$ **and**

to-set-wrap [*simp*]: $\wedge a. \text{to-set } (\text{wrap } a) = \{a\}$

begin

lemma *to-set-add* [*simp*]: $\text{to-set } (\text{add } a b) = \text{insert } a (\text{to-set } b)$

<proof>

end

locale *natural-magma-with-empty* = *natural-magma* +

fixes *empty*

assumes *to-set-empty* [*simp*]: $\text{to-set } \text{empty} = \{\}$

end

theory *Natural-Functor*

imports *Main*

begin

locale *natural-functor* =

fixes

```

    map :: ('a ⇒ 'a) ⇒ 'b ⇒ 'b and
    to-set :: 'b ⇒ 'a set
assumes
    map-comp [simp]:  $\bigwedge b f g. \text{map } f (\text{map } g b) = \text{map } (\lambda x. f (g x)) b$  and
    map-ident [simp]:  $\bigwedge b. \text{map } (\lambda x. x) b = b$  and
    map-cong0 [cong]:  $\bigwedge b f g. (\bigwedge a. a \in \text{to-set } b \implies f a = g a) \implies \text{map } f b = \text{map } g b$  and
    to-set-map [simp]:  $\bigwedge b f. \text{to-set } (\text{map } f b) = f \text{ ' to-set } b$  and
    exists-functor [intro]:  $\bigwedge a. \exists b. a \in \text{to-set } b$ 
begin

lemma map-id [simp]: map id b = b
  <proof>

lemma map-cong [cong]:
  assumes  $b = b' \bigwedge a. a \in \text{to-set } b' \implies f a = g a$ 
  shows map f b = map g b'
  <proof>

end

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

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

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

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

```

```

begin

lemma map-add:  $\bigwedge f a b. \text{map } f (\text{add } a b) = \text{add } (f a) (\text{map } f b)$ 
  <proof>

end

end
theory Functional-Substitution-Lifting
  imports Functional-Substitution Natural-Magma-Function
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  $\equiv \bigcup (\text{sub-vars } ' \text{to-set } \text{expr})$ 

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

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

lemma map-id-cong [simp]:
  assumes  $\bigwedge \text{sub}. \text{sub} \in \text{to-set } \text{expr} \implies f \text{sub} = \text{sub}$ 
  shows map f expr = expr
  <proof>

lemma to-set-map-not-ident:
  assumes sub  $\in \text{to-set } \text{expr}$  f sub  $\notin \text{to-set } \text{expr}$ 
  shows map f expr  $\neq$  expr
  <proof>

lemma subst-in-to-set-subst [intro]:
  assumes sub  $\in \text{to-set } \text{expr}$ 
  shows sub  $\cdot_s \sigma \in \text{to-set } (\text{expr } \cdot \sigma)$ 
  <proof>

sublocale functional-substitution where subst = ( $\cdot$ ) and vars = vars
  <proof>

lemma ground-subst-iff-sub-ground-subst [simp]: is-ground-subst  $\gamma \longleftrightarrow \text{sub.is-ground-subst}$ 

```


γ
 $\langle proof \rangle$

lemma *to-set-is-ground* [intro]:
 assumes $sub \in to\text{-}set\ expr\ is\text{-}ground\ expr$
 shows $sub.is\text{-}ground\ sub$
 $\langle proof \rangle$

lemma *to-set-is-ground-subst*:
 assumes $sub \in to\text{-}set\ expr\ is\text{-}ground\ (expr \cdot \gamma)$
 shows $sub.is\text{-}ground\ (sub \cdot_s \gamma)$
 $\langle proof \rangle$

lemma *subst-empty*:
 assumes $to\text{-}set\ expr' = \{\}$
 shows $expr \cdot \sigma = expr' \longleftrightarrow expr = expr'$
 $\langle proof \rangle$

lemma *empty-is-ground*:
 assumes $to\text{-}set\ expr = \{\}$
 shows $is\text{-}ground\ expr$
 $\langle proof \rangle$

lemma *to-set-image*: $to\text{-}set\ (expr \cdot \sigma) = (\lambda a. a \cdot_s \sigma) ` to\text{-}set\ expr$
 $\langle proof \rangle$

lemma *to-set-subset-vars-subset*:
 assumes $to\text{-}set\ expr \subseteq to\text{-}set\ expr'$
 shows $vars\ expr \subseteq vars\ expr'$
 $\langle proof \rangle$

lemma *to-set-subset-is-ground*:
 assumes $to\text{-}set\ expr' \subseteq to\text{-}set\ expr\ is\text{-}ground\ expr$
 shows $is\text{-}ground\ expr'$
 $\langle proof \rangle$

end

locale *based-functional-substitution-lifting* =
 functional-substitution-lifting +
 base: *base-functional-substitution* **where** $subst = base\text{-}subst$ **and** $vars = base\text{-}vars$
 +
 sub: *based-functional-substitution* **where** $subst = sub\text{-}subst$ **and** $vars = sub\text{-}vars$
begin

sublocale *based-functional-substitution* **where** $subst = subst$ **and** $vars = vars$
 $\langle proof \rangle$

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*
⟨*proof*⟩

lemma *fset-to-fset* [*simp*]: *fset* (*to-fset expr*) = *to-set expr*
⟨*proof*⟩

lemma *to-fset-map*: *to-fset* (*map f expr*) = *f* |*q*| *to-fset expr*
⟨*proof*⟩

lemma *to-fset-is-ground-subst*:
assumes *sub* |*∈*| *to-fset expr is-ground* (*subst expr* *γ*)
shows *sub.is-ground* (*sub* ·_{*s*} *γ*)
⟨*proof*⟩

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*⟩

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*⟩

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* ⇒ '*sub*_G **and**
 sub-from-ground :: '*sub*_G ⇒ '*sub* **and**
 sub-vars :: '*sub* ⇒ '*var set* **and**
 sub-subst :: '*sub* ⇒ ('*var* ⇒ '*base*) ⇒ '*sub* **and**
 map :: ('*sub* ⇒ '*sub*) ⇒ '*expr* ⇒ '*expr* **and**
 to-ground-map :: ('*sub* ⇒ '*sub*_G) ⇒ '*expr* ⇒ '*expr*_G **and**
 from-ground-map :: ('*sub*_G ⇒ '*sub*) ⇒ '*expr*_G ⇒ '*expr* **and**
 ground-map :: ('*sub*_G ⇒ '*sub*_G) ⇒ '*expr*_G ⇒ '*expr*_G **and**
 to-set-ground :: '*expr*_G ⇒ '*sub*_G *set*

begin

definition *to-ground* :: '*expr* ⇒ '*expr*_G **where**
 to-ground *expr* ≡ *to-ground-map* *sub-to-ground* *expr*

definition *from-ground* :: '*expr*_G ⇒ '*expr* **where**
 from-ground *expr*_G ≡ *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*⟩

lemma *to-set-from-ground*: *to-set* (*from-ground* *expr*) = *sub-from-ground* ' (*to-set-ground*
expr)
 ⟨*proof*⟩

lemma *sub-in-ground-is-ground*:
 assumes *sub* ∈ *to-set* (*from-ground* *expr*)
 shows *sub.is-ground* *sub*
 ⟨*proof*⟩

lemma *ground-sub-in-ground*:
 sub ∈ *to-set-ground* *expr* ⟷ *sub-from-ground* *sub* ∈ *to-set* (*from-ground* *expr*)
 ⟨*proof*⟩

lemma *ground-sub*:

$(\forall sub \in to\text{-}set (from\text{-}ground\ expr_G). P\ sub) \longleftrightarrow$
 $(\forall sub_G \in to\text{-}set\text{-}ground\ expr_G. P (sub\text{-}from\text{-}ground\ sub_G))$
 $\langle proof \rangle$

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* \Rightarrow '*sub*) \Rightarrow '*expr* \Rightarrow '*expr*
begin

sublocale *all-subst-ident-iff-ground* **where** *subst* = *subst* **and** *vars* = *vars*
 $\langle proof \rangle$

end

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

lemma *vars-add* [*simp*]:
 $vars (add\ sub\ expr) = sub\text{-}vars\ sub \cup vars\ expr$
 $\langle proof \rangle$

lemma *vars-plus* [*simp*]:
 $vars (plus\ expr\ expr') = vars\ expr \cup vars\ expr'$
 $\langle proof \rangle$

lemma *is-ground-add* [*simp*]:
 $is\text{-}ground (add\ sub\ expr) \longleftrightarrow sub.is\text{-}ground\ sub \wedge is\text{-}ground\ expr$
 $\langle proof \rangle$

end

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

lemma *add-subst* [*simp*]:
 $(add\ sub\ expr) \cdot \sigma = add (sub \cdot_s \sigma) (expr \cdot \sigma)$
 $\langle proof \rangle$

lemma *plus-subst* [*simp*]: $(plus\ expr\ expr') \cdot \sigma = plus (expr \cdot \sigma) (expr' \cdot \sigma)$
 $\langle proof \rangle$

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]:*
 $from_ground\ (plus_ground\ expr\ expr') = plus\ (from_ground\ expr)\ (from_ground\ expr')$
<proof>

lemma *from-ground-add [simp]:*
 $from_ground\ (add_ground\ sub\ expr) = add\ (sub_from_ground\ sub)\ (from_ground\ expr)$
<proof>

lemma *to-ground-add [simp]:*
 $to_ground\ (add\ sub\ expr) = add_ground\ (sub_to_ground\ sub)\ (to_ground\ expr)$
<proof>

lemma *ground-add:*
assumes *from-ground expr = add sub expr'*
shows *expr = add-ground (sub-to-ground sub) (to-ground expr')*
<proof>

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
  ⟨proof⟩

lemma to-ground-empty' [simp]: to-ground expr = empty-ground  $\longleftrightarrow$  expr =
  empty
  ⟨proof⟩

lemma from-ground-empty' [simp]: from-ground expr = empty  $\longleftrightarrow$  expr = empty-ground
  ⟨proof⟩

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  $\times$  ('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  $\Rightarrow$  '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 =  $\lambda f$ . map-prod
  ff and
  to-set = set-prod
  ⟨proof⟩

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*

<proof>

Lifted lemmas and definitions

thm

equation-set-subst.subst-redundant-upd

equation-set-subst.subst-redundant-if

equation-set-subst.vars-subst-subset

equation-set-subst.vars-def

equation-set-subst.subst-def

end