

# Abstract Substitutions as Monoid Actions

Martin Desharnais

April 11, 2026

## 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

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>General Results on Groups</b>              | <b>1</b> |
| <b>2</b> | <b>Monoid</b>                                 | <b>2</b> |
| <b>3</b> | <b>Semigroup Action</b>                       | <b>2</b> |
| <b>4</b> | <b>Monoid Action</b>                          | <b>3</b> |
| <b>5</b> | <b>Group Action</b>                           | <b>4</b> |
| <b>6</b> | <b>Assumption-free Substitution</b>           | <b>4</b> |
| <b>7</b> | <b>Basic Substitution</b>                     | <b>7</b> |
| 7.1      | Substitution Composition . . . . .            | 8        |
| 7.2      | Substitution Identity . . . . .               | 9        |
| 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 . . . . .           | 12       |
| 7.12     | Instances of Renamed Expressions . . . . .    | 12       |

|           |   |           |
|-----------|---|-----------|
| <b>8</b>  | <b>Substitutions on variables</b>                           | <b>15</b> |
| 8.1       | Properties of substitutions . . . . .                       | 16        |
| <b>9</b>  | <b>Substitutions on base expressions</b>                    | <b>23</b> |
| <b>10</b> | <b>Properties of substitutions on base expressions</b>      | <b>25</b> |
| <b>11</b> | <b>Lifting of substitutions using natural functors</b>      | <b>30</b> |
| 11.1      | Lifting of properties . . . . .                             | 32        |
| <b>12</b> | <b>Lifting of based substitutions</b>                       | <b>38</b> |
| 12.1      | Lifting of properties . . . . .                             | 38        |
| <b>13</b> | <b>Substitutions for first order terms</b>                  | <b>41</b> |
| 13.1      | Interpretations for first order terms . . . . .             | 41        |
| 13.2      | Compatibility with First_Order_Term . . . . .               | 42        |
| 13.3      | Interpretations for IMGUs . . . . .                         | 43        |
| 13.4      | Additional lemmas . . . . .                                 | 43        |
| <b>14</b> | <b>Substitutions for first order terms as binary tree</b>   | <b>43</b> |
| 14.1      | Substitution monoid . . . . .                               | 43        |
| 14.2      | Transfer definitions and lemmas from HOL-ex.Unification . . | 44        |
| 14.3      | Base Substitution . . . . .                                 | 44        |
| 14.4      | Substitution Properties . . . . .                           | 45        |
| 14.5      | Compatibility with HOL-ex.Unification . . . . .             | 45        |
| 14.6      | Interpretations for IMGUs . . . . .                         | 46        |

```

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  $\bigwedge a. inv\ (inv\ a) = a$ 
  <proof>

```

```

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

```

```

lemma (in monoid) group-wrt-right-inverse:
  fixes inv

```

**assumes** *right-inverse*:  $\bigwedge a. a * \text{inv } a = \mathbf{1}$   
**shows** *group*  $(*) \mathbf{1} \text{ inv}$   
*<proof>*

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

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

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

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

**lemma** (*in monoid*) *neutral-is-left-invertible* [*simp*]:  
*is-left-invertible*  $\mathbf{1}$   
*<proof>*

**lemma** (*in monoid*) *neutral-is-right-invertible* [*simp*]:  
*is-right-invertible*  $\mathbf{1}$   
*<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  $\Rightarrow$  'b  $\Rightarrow$  'b (**infix**  $\langle \cdot \rangle$  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**  $\langle \cdot \rangle$  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**  $action :: 'a \Rightarrow 'b \Rightarrow 'b$  (**infix**  $\langle \cdot \rangle$  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**  $\langle \cdot \rangle$  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* (\*)  $\mathbf{1}$  ( $\lambda X a. (\lambda x. action x a) ' X$ )

*<proof>*

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

## 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*  $\subseteq$  *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* =  $\lambda x\ y.\ \text{action}\ y\ x$   
*<proof>*

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

**end**

**theory** *Abstract-Substitution*

**imports** *Monoid-Action*

**begin**

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

## 6 Assumption-free Substitution

**locale** *abstract-substitution-ops* =  
**fixes**  
*subst* :: 'x  $\Rightarrow$  's  $\Rightarrow$  'x (**infixl** <·> 67) **and**  
*id-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**  $instances :: 'x \Rightarrow 'x\ set$  **where**  
 $instances\ x = \{y. generalizes\ x\ y\}$

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

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

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

**lemma**  $ground\text{-}instances\text{-}set\text{-}eq\text{-}Union\text{-}ground\text{-}instances$ :  
 $ground\text{-}instances\text{-}set\ X = (\bigcup x \in X. ground\text{-}instances\ x)$   
*<proof>*

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

**lemma**  $mem\text{-}ground\text{-}instancesE[elim]$ :

**fixes**  $x x_G :: 'x$   
**assumes**  $x_G \in \text{ground-instances } x$   
**obtains**  $\gamma :: 's$  **where**  $x_G = x \cdot \gamma$  **and**  $\text{is-ground } (x \cdot \gamma)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{mem-ground-instances-setE}[\text{elim}]$ :  
**fixes**  $x_G :: 'x$  **and**  $X :: 'x \text{ set}$   
**assumes**  $x_G \in \text{ground-instances-set } X$   
**obtains**  $x :: 'x$  **and**  $\gamma :: 's$  **where**  $x \in X$  **and**  $x_G = x \cdot \gamma$  **and**  $\text{is-ground } (x \cdot \gamma)$   
 $\langle \text{proof} \rangle$

**definition**  $\text{is-unifier} :: 's \Rightarrow 'x \text{ set} \Rightarrow \text{bool}$  **where**  
 $\text{is-unifier } v X \longleftrightarrow X = \{\} \vee \text{card } (\text{subst-set } X v) = 1$

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

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

**definition**  $\text{is-imgu} :: 's \Rightarrow 'x \text{ set set} \Rightarrow \text{bool}$  **where**  
 $\text{is-imgu } \mu XX \longleftrightarrow \text{is-unifier-set } \mu XX \wedge (\forall \tau. \text{is-unifier-set } \tau XX \longrightarrow \mu \odot \tau = \tau)$

**lemma**  $\text{is-unifier-iff}$ :  $\text{is-unifier } \sigma X \longleftrightarrow (\forall x \in X. \forall y \in X. x \cdot \sigma = y \cdot \sigma)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-unifier-singleton}[\text{simp}]$ :  $\text{is-unifier } v \{x\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-unifier-empty}[\text{simp}]$ :  $\text{is-unifier } \sigma \{\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-unifier-set-empty}[\text{simp}]$ :  $\text{is-unifier-set } \sigma \{\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-unifier-set-insert}$ :  
 $\text{is-unifier-set } \sigma (\text{insert } X XX) \longleftrightarrow \text{is-unifier } \sigma X \wedge \text{is-unifier-set } \sigma XX$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-unifier-set-insert-singleton}[\text{simp}]$ :  
 $\text{is-unifier-set } \sigma (\text{insert } \{x\} XX) \longleftrightarrow \text{is-unifier-set } \sigma XX$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-mgu-insert-singleton}[\text{simp}]$ :  $\text{is-mgu } \mu (\text{insert } \{x\} XX) \longleftrightarrow \text{is-mgu } \mu XX$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{is-imgu-insert-singleton}[\text{simp}]$ :  $\text{is-imgu } \mu (\text{insert } \{x\} XX) \longleftrightarrow \text{is-imgu } \mu$

*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  $X$   $\sigma$ )*  
*<proof>*

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

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

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

**lemma** *subst-list-append[simp]: subst-list ( $xs_1 @ xs_2$ )  $\sigma$  = subst-list  $xs_1$   $\sigma @$  subst-list  $xs_2$   $\sigma$*   
*<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  $v$   $A \implies B \subseteq A \implies$  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** *abstract-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*

*<proof>*

**lemma** *exists-renaming* [*intro*]:  $\exists \rho. \textit{is-renaming } \rho$

*<proof>*

**lemma** *exists-idem* [*intro*]:  $\exists \sigma. \textit{is-idem } \sigma$

*<proof>*

**end**

**locale** *abstract-substitution* =

*abstract-substitution-monoid comp-subst id-subst +*  
*comp-subst: right-monoid-action comp-subst id-subst subst +*  
*abstract-substitution-ops subst id-subst comp-subst is-ground*

**for**

*comp-subst* :: 's  $\Rightarrow$  's  $\Rightarrow$  's (**infixl**  $\langle \odot \rangle$  70) **and**

*id-subst* :: 's **and**

*subst* :: 'x  $\Rightarrow$  's  $\Rightarrow$  'x (**infixl**  $\langle \cdot \rangle$  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*

*<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-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  $\longleftrightarrow$  X = {}  $\vee$  card X = 1*  
*<proof>*

**lemma** *is-unifier-set-id-subst: is-unifier-set id-subst XX  $\longleftrightarrow$  ( $\forall X \in XX. X = {}$   
 $\vee$  card X = 1)*  
*<proof>*

**lemma** *is-mgu-id-subst: is-mgu id-subst XX  $\longleftrightarrow$  ( $\forall X \in XX. X = {}$   
 $\vee$  card X = 1)*  
*<proof>*

**lemma** *is-imgu-id-subst: is-imgu id-subst XX  $\longleftrightarrow$  ( $\forall X \in XX. X = {}$   
 $\vee$  card X = 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]*:  $is\_ground\_set\ X \implies subst\_set\ X\ \sigma = X$   
*<proof>*

## 7.5 Instances of Ground Expressions

**lemma** *instances-ident-if-ground[simp]*:  $is\_ground\ x \implies instances\ x = \{x\}$   
*<proof>*

**lemma** *instances-set-ident-if-ground[simp]*:  $is\_ground\_set\ X \implies instances\_set\ X = X$   
*<proof>*

**lemma** *ground-instances-ident-if-ground[simp]*:  $is\_ground\ x \implies ground\_instances\ x = \{x\}$   
*<proof>*

**lemma** *ground-instances-set-ident-if-ground[simp]*:  $is\_ground\_set\ X \implies ground\_instances\_set\ X = X$   
*<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>*

**corollary** *ground-eq-ground-if-mgu*:  
assumes  $is\_mgu\ \mu\ \{\{t_1, t_2\}\}$   $is\_ground\ t_1$   $is\_ground\ t_2$   
shows  $t_1 = t_2$   
*<proof>*

**corollary** *ground-eq-ground-if-imgu*:  
assumes  $is\_imgu\ \mu\ \{\{t_1, t_2\}\}$   $is\_ground\ t_1$   $is\_ground\ t_2$   
shows  $t_1 = t_2$   
*<proof>*

**lemma** *ball-eq-constant-if-unifier*:  
assumes  $x \in X$  and  $is\_unifier\ v\ X$  and  $is\_ground\_set\ X$   
shows  $\forall y \in X. y = x$   
*<proof>*

**lemma** *is-mgu-unifies*:  
assumes  $is\_mgu\ \mu\ XX$   
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*  $\mu$   $XX$

**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*  $\gamma$

**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*  $\mu$   $\{X\}$  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*  $X$   $\sigma$ )  $\subseteq$  *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*  $X$   $\sigma$ )  $\subseteq$  *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*  $X$   $\varrho$ ) = *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*  $\varrho \implies$  *ground-instances-set* (*subst-set*  $X$   $\varrho$ ) = *ground-instances-set*  $X$   
(*proof*)

**end**

**end**

**theory** *Option-Extra*

**imports** *Main*

**begin**

**abbreviation** *get-or* :: 'a option  $\Rightarrow$  'a  $\Rightarrow$  'a **where**  
 $\wedge$  a default. *get-or* a default  $\equiv$  case a of None  $\Rightarrow$  default | Some a  $\Rightarrow$  a

**end**

**theory** *List-Extra*  
**imports** *Main*  
**begin**

**definition** *set-list* :: 'a set  $\Rightarrow$  'a list **where**  
*set-list* S  $\equiv$  SOME xs. set xs = S  $\wedge$  distinct xs

**lemma** *set-list-empty* [simp]: *set-list* {} = []  
<proof>

**lemma** *set-list-singleton* [simp]: *set-list* {x} = [x]  
<proof>

**lemma** *set-list-set* [simp]:  
**assumes** finite S  
**shows** set (*set-list* S) = S  
<proof>

**lemma** *set-list-distinct* [simp]:  
**assumes** finite S  
**shows** distinct (*set-list* S)  
<proof>

**end**

**theory** *Finite-Map-Extra*  
**imports**  
*HOL-Library.Finite-Map*  
*List-Extra*  
**begin**

**definition** *fmap-dom-list* :: ('k, 'v) fmap  $\Rightarrow$  'k list **where**  
*fmap-dom-list* m  $\equiv$  SOME xs. set xs = fmdom' m  $\wedge$  distinct xs

**lemma** *fmap-dom-list-exists* [intro]:  $\exists$  xs. set xs = fmdom' m  $\wedge$  distinct xs  
<proof>

**lemma** *set-fmap-dom-list* [simp]: set (*fmap-dom-list* m) = fmdom' m  
<proof>

**lemma** *distinct-fmap-dom-list* [simp]: distinct (*fmap-dom-list* m)  
<proof>

**lemma** *fmap-dom-list-empty* [simp]: *fmap-dom-list* fmempty = []  
<proof>

**definition** *fmap-list* :: ('k, 'v) fmap  $\Rightarrow$  ('k  $\times$  'v) list **where**  
*fmap-list* m  $\equiv$  map ( $\lambda k. (k, \text{the } (\text{fmlookup } m \ k))$ ) (fmap-dom-list m)

**lemma** *fmap-list-empty* [simp]: *fmap-list* fmempty = []  
(proof)

**lemma** *set-fst-fmap-list* [simp]: set (map fst (fmap-list m)) = fmdom' m  
(proof)

**lemma** *distinct-fst-fmap-list* [simp]: distinct (map fst (fmap-list m))  
(proof)

**lemma** *fmap-list-mem-iff*: (k, v)  $\in$  set (fmap-list m)  $\longleftrightarrow$  fmlookup m k = Some v  
(proof)

**definition** *fmap-of-set* :: 'k set  $\Rightarrow$  ('k  $\Rightarrow$  'v)  $\Rightarrow$  ('k, 'v) fmap **where**  
*fmap-of-set* X f  $\equiv$  fold ( $\lambda x \ m. \text{fmupd } x \ (f \ x) \ m$ ) (set-list X) fmempty

**lemma** *fmap-of-set-empty* [simp]: *fmap-of-set* {} f = fmempty  
(proof)

**lemma** *fmlookup-fold-fmupd-notin* [simp]:  
assumes  $x \notin$  set xs  
shows fmlookup (fold ( $\lambda k \ m. \text{fmupd } k \ (f \ k) \ m$ ) xs m) x = fmlookup m x  
(proof)

**lemma** *fmlookup-fold-fmupd-in* [simp]:  
assumes distinct xs  $x \in$  set xs  
shows fmlookup (fold ( $\lambda k \ m. \text{fmupd } k \ (f \ k) \ m$ ) xs m) x = Some (f x)  
(proof)

**lemma** *fmlookup-fmap-of-set-notin* [simp]:  
assumes finite X  $x \notin$  X  
shows fmlookup (fmap-of-set X f) x = None  
(proof)

**lemma** *fmlookup-fmap-of-set-in* [simp]:  
assumes finite X **and**  $x \in$  X  
shows fmlookup (fmap-of-set X f) x = Some (f x)  
(proof)

**end**

**theory** *Fun-Extra*

imports *Main*

**begin**

**definition** *bij-partition* **where**  
*bij-partition* S T  $\equiv$  SOME h. *bij-betw* h (T - S) (S - T)

**lemma** *bij-partition*:  
**assumes** *finite S finite T card S = card T*  
**shows** *bij-betw (bij-partition S T) (T - S) (S - T)*  
*<proof>*

**end**

**theory** *Substitution*

**imports**

*Abstract-Substitution*

*Option-Extra*

*Finite-Map-Extra*

*Fun-Extra*

**begin**

## 8 Substitutions on variables

**locale** *substitution = abstract-substitution where*

*subst = subst*

**for**

*subst* :: *'expr ⇒ 'subst ⇒ 'expr (infixl · 69)* **and**

*apply-subst* :: *'v ⇒ 'subst ⇒ 'base (infixl ·v 69)* **and**

*vars* :: *'expr ⇒ 'v set +*

**assumes** *no-vars-if-is-ground [intro]: ∧ expr. is-ground expr ⇒ vars expr = {}*

**begin**

**abbreviation** *exists-nonground where*

*exists-nonground ≡ ∃ expr. ¬is-ground expr*

**definition** *vars-set* :: *'expr set ⇒ 'v set where*

*vars-set exprs ≡ ∪ expr ∈ exprs. vars expr*

**lemma** *subst-cannot-unground*:

**assumes** *¬is-ground (expr · σ)*

**shows** *¬is-ground expr*

*<proof>*

**abbreviation** (*input*) *var-subst where*

*var-subst σ x ≡ x ·v σ*

**abbreviation** (*input*) *expr-subst where*

*expr-subst σ expr ≡ expr · σ*

**definition** *subst-domain* :: *'subst ⇒ 'v set where*

*subst-domain σ = {x. x ·v σ ≠ x ·v id-subst}*

**abbreviation** *subst-range* :: *'subst ⇒ 'base set where*

*subst-range σ ≡ var-subst σ ` subst-domain σ*

**lemma** *subst-inv*:

**assumes**  $\sigma \odot \sigma\text{-inv} = \text{id-subst}$   
**shows**  $\text{expr} \cdot \sigma \cdot \sigma\text{-inv} = \text{expr}$   
 $\langle \text{proof} \rangle$

**definition** *rename* **where**

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

**lemma** *is-unifier-two-iff* [simp]: *is-unifier*  $v \ \{ \text{expr}, \text{expr}' \} \longleftrightarrow \text{expr} \cdot v = \text{expr}' \cdot v$   
 $\langle \text{proof} \rangle$

**lemma** *is-unifier-set-two-iff* [simp]: *is-unifier-set*  $v \ \{ \{ \text{expr}, \text{expr}' \} \} \longleftrightarrow \text{expr} \cdot v = \text{expr}' \cdot v$   
 $\langle \text{proof} \rangle$

**lemma** *obtain-imgu-absorption*:

**assumes** *is-unifier-set*  $v \ XX$  *is-imgu*  $\mu \ XX$

**obtains**  $\sigma$  **where**  $v = \mu \odot \sigma$

$\langle \text{proof} \rangle$

**end**

## 8.1 Properties of substitutions

**locale** *subst-update-def* =

**fixes** *subst-update* ::  $'\text{subst} \Rightarrow 'v \Rightarrow 'base \Rightarrow 'subst$

**begin**

**definition** *subst-updates* ::  $'\text{subst} \Rightarrow ('v, 'base) \text{fmap} \Rightarrow 'subst \ (\langle \text{[-]} \rangle [1000, 0] \ 71)$

**where**

*subst-updates*  $\sigma \ m = \text{fold} \ (\lambda(x, b) \ \sigma. \ \text{subst-update } \sigma \ x \ b) \ (\text{fmap-list } m) \ \sigma$

**end**

**locale** *subst-update* =

*substitution* **where** *vars* = *vars* **and** *apply-subst* = *apply-subst* +

*subst-update-def* **where** *subst-update* = *subst-update*

**for**

*vars* ::  $'\text{expr} \Rightarrow 'v \ \text{set}$  **and**

*apply-subst* ::  $'v \Rightarrow 'subst \Rightarrow 'base \ (\text{infixl } \cdot v \ 69)$  **and**

*subst-update* ::  $'\text{subst} \Rightarrow 'v \Rightarrow 'base \Rightarrow 'subst \ (\langle \text{[-] := -} \rangle [1000, 0, 50] \ 71) +$

**assumes**

*subst-update-var* [simp]:

— The precondition of the assumption ensures noop substitutions

$\bigwedge x \ u \ \sigma. \ \text{exists-nonground} \implies x \cdot v \ \sigma[x := u] = u$

$\bigwedge x \ y \ u \ \sigma. \ x \neq y \implies x \cdot v \ \sigma[y := u] = x \cdot v \ \sigma$  **and**

*subst-update* [simp]:

$\bigwedge x \ \text{expr} \ u \ \sigma. \ x \notin \text{vars } \text{expr} \implies \text{expr} \cdot \sigma[x := u] = \text{expr} \cdot \sigma$

$\bigwedge \sigma \ x. \ \sigma[x := x \cdot v \ \sigma] = \sigma$  **and**

*subst-update-twice* [simp]:  
 $\bigwedge \sigma \ x \ a \ b. (\sigma[x := a])[x := b] = \sigma[x := b]$   
**begin**

**lemma** *subst-updates-empty* [simp]:  $\sigma[fmempty] = \sigma$   
 ⟨proof⟩

**lemma** *fold-redundant-updates-var* [simp]:  
**assumes**  $x \notin \text{set } (\text{map } \text{fst } us)$   
**shows**  $x \cdot v \text{ fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us \ \sigma = x \cdot v \ \sigma$   
 ⟨proof⟩

**lemma** *fold-updates-var* [simp]:  
**assumes**  
*exists-nonground*: *exists-nonground* **and**  
*distinct-updates*: *distinct* (map fst us) **and**  
*update-in-updates*:  $(x, b) \in \text{set } us$   
**shows**  $x \cdot v \text{ fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us \ \sigma = b$   
 ⟨proof⟩

**lemma** *fold-redundant-updates* [simp]:  
**assumes**  $\bigwedge x \ b. (x, b) \in \text{set } us \implies x \notin \text{vars } \text{expr} \vee b = x \cdot v \ \sigma \text{ distinct } (\text{map } \text{fst } us)$   
**shows**  $\text{expr} \cdot \text{fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us \ \sigma = \text{expr} \cdot \sigma$   
 ⟨proof⟩

**lemma** *subst-updates-var*:  
**assumes** *exists-nonground*: *exists-nonground*  
**shows**  $x \cdot v \ \sigma[u] = \text{get-or } (\text{fmlookup } u \ x) \ (x \cdot v \ \sigma)$   
 ⟨proof⟩

**lemma** *redundant-subst-updates* [simp]:  
**assumes**  $\bigwedge x. x \in \text{vars } \text{expr} \implies \text{fmlookup } u \ x = \text{None} \vee \text{fmlookup } u \ x = \text{Some } (x \cdot v \ \sigma)$   
**shows**  $\text{expr} \cdot \sigma[u] = \text{expr} \cdot \sigma$   
 ⟨proof⟩

**lemma** *redundant-subst-updates-vars-set* [simp]:  
**assumes** *exists-nonground*  $\bigwedge x. x \in X \implies \text{fmlookup } u \ x = \text{None} \vee \text{fmlookup } u \ x = \text{Some } (x \cdot v \ \sigma)$   
**shows**  $(\lambda x. x \cdot v \ \sigma[u]) \ ' X = (\lambda x. x \cdot v \ \sigma) \ ' X$   
 ⟨proof⟩

**lemma** *redundant-subst-updates-vars-image* [simp]:  
**assumes**  $\bigwedge x. x \in \bigcup (\text{vars } \ ' X) \implies \text{fmlookup } u \ x = \text{None} \vee \text{fmlookup } u \ x = \text{Some } (x \cdot v \ \sigma)$   
**shows**  $(\lambda \text{expr}. \text{expr} \cdot \sigma[u]) \ ' X = (\lambda \text{expr}. \text{expr} \cdot \sigma) \ ' X$   
 ⟨proof⟩

**lemma** *subst-updates-fmap-of-set* [*simp*]:  
**assumes** *exists-nonground*  $x \in X$  *finite*  $X$   
**shows**  $x \cdot v \sigma \llbracket \text{fmap-of-set } X f \rrbracket = f x$   
*<proof>*

**lemma** *redundant-subst-updates-fmap-of-set* [*simp*]:  
**assumes** *exists-nonground*  $x \notin X$  *finite*  $X$   
**shows**  $x \cdot v \sigma \llbracket \text{fmap-of-set } X f \rrbracket = x \cdot v \sigma$   
*<proof>*

**definition** *renaming-of-bij* **where**  
*renaming-of-bij*  $f S T \equiv$   
 $\text{id-subst} \llbracket \text{fmap-of-set } (S \cup T) (\lambda x. (\text{if } x \in S \text{ then } f x \text{ else } \text{bij-partition } S T x)) \cdot v$   
 $\text{id-subst} \rrbracket$

**definition** *renaming-of-bij-inv* **where**  
*renaming-of-bij-inv*  $f S T \equiv$   
 $\text{id-subst} \llbracket \text{fmap-of-set } (S \cup T)$   
 $(\lambda x. (\text{if } x \in T \text{ then } \text{inv-into } S f x \text{ else } \text{inv-into } (T - S) (\text{bij-partition } S T) x))$   
 $\cdot v \text{id-subst} \rrbracket$

**lemma** *redundant-renaming-of-bij*:  
**assumes** *exists-nonground* *finite*  $S$  *bij-betw*  $f S T$   $x \notin S \cup T$   
**shows**  $x \cdot v \text{renaming-of-bij } f S T = x \cdot v \text{id-subst}$   
*<proof>*

**lemma** *renaming-of-bij-on-S*:  
**assumes** *exists-nonground* *finite*  $S$  *bij-betw*  $f S T$   $x \in S$   
**shows**  $x \cdot v \text{renaming-of-bij } f S T = f x \cdot v \text{id-subst}$   
*<proof>*

**end**

**locale** *all-subst-ident-iff-ground* =  
*substitution* +  
**assumes** *all-subst-ident-iff-ground*:  $\bigwedge \text{expr}. \text{is-ground } \text{expr} \iff (\forall \sigma. \text{expr} \cdot \sigma = \text{expr})$

**locale** *exists-non-ident-subst* =  
*substitution* **where** *vars* = *vars*  
**for** *vars* ::  $'\text{expr} \Rightarrow 'v \text{ set}$  +  
**assumes**  
*exists-non-ident-subst*:  
 $\bigwedge \text{expr } S. \text{finite } S \implies \neg \text{is-ground } \text{expr} \implies \exists \sigma. \text{expr} \cdot \sigma \neq \text{expr} \wedge \text{expr} \cdot \sigma \notin S$

**begin**

**lemma** *infinite-exprs*:  
**assumes** *exists-nonground*

**shows** *infinite* (*UNIV* :: 'expr set)  
 ⟨*proof*⟩

**end**

**locale** *finite-variables = substitution* **where** *vars = vars*  
**for** *vars* :: 'expr ⇒ 'v set +  
**assumes** *finite-vars* [*intro*]:  $\bigwedge \text{expr}. \text{finite } (\text{vars expr})$   
**begin**

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

**lemma** *fset-finite-vars* [*simp*]:  $\text{fset } (\text{finite-vars expr}) = \text{vars expr}$   
 ⟨*proof*⟩

**end**

**locale** *infinite-variables = substitution* **where** *vars = vars*  
**for** *vars* :: 'expr ⇒ 'v set +  
**assumes** *infinite-vars* [*intro*]: *infinite (UNIV :: 'v set)*

**locale** *renaming-variables = substitution +*  
**assumes**  
 — The precondition of the assumption ensures noop substitutions  
*is-renaming-imp*:  
 $\bigwedge \varrho. \text{exists-nonground } \varrho \implies$   
 $\text{is-renaming } \varrho \implies \text{inj } (\text{var-subst } \varrho) \wedge (\forall x. \exists x'. x \cdot v \varrho = x' \cdot v \text{id-subst})$

**and**  
*rename-variables*:  $\bigwedge \text{expr } \varrho. \text{is-renaming } \varrho \implies \text{vars } (\text{expr} \cdot \varrho) = \text{rename } \varrho \text{ `}$   
 (*vars expr*)

**begin**

**lemma** *renaming-range-id-subst*:  
**assumes** *exists-nonground*: *exists-nonground* **and**  $\varrho$ : *is-renaming*  $\varrho$   
**shows**  $x \cdot v \varrho \in \text{range } (\text{var-subst id-subst})$   
 ⟨*proof*⟩

**lemma** *obtain-renamed-variable*:  
**assumes** *exists-nonground* *is-renaming*  $\varrho$   
**obtains**  $x'$  **where**  $x \cdot v \varrho = x' \cdot v \text{id-subst}$   
 ⟨*proof*⟩

**lemma** *id-subst-rename* [*simp*]:  
**assumes** *exists-nonground* **and**  $\varrho$ : *is-renaming*  $\varrho$   
**shows**  $\text{rename } \varrho \ x \cdot v \text{id-subst} = x \cdot v \varrho$   
 ⟨*proof*⟩

**lemma** *rename-variables-id-subst*:

**assumes** *is-renaming*  $\varrho$

**shows**  $\text{var-subst } id\text{-subst } \cdot \text{vars } (expr \cdot \varrho) = \text{var-subst } \varrho \cdot (\text{vars } expr)$

*<proof>*

**lemma** *surj-inv-renaming*:

**assumes** *exists-nonground*: *exists-nonground* **and**  $\varrho$ : *is-renaming*  $\varrho$

**shows** *surj*  $(\lambda x. \text{inv } (\text{var-subst } \varrho) (x \cdot v \text{ id-subst}))$

*<proof>*

**lemma** *renaming-range*:

**assumes**  $\varrho$ : *is-renaming*  $\varrho$  **and**  $x: x \in \text{vars } (expr \cdot \varrho)$

**shows**  $x \cdot v \text{ id-subst} \in \text{range } (\text{var-subst } \varrho)$

*<proof>*

**lemma** *renaming-inv-into*:

**assumes** *is-renaming*  $\varrho$   $x \in \text{vars } (expr \cdot \varrho)$

**shows**  $\text{inv } (\text{var-subst } \varrho) (x \cdot v \text{ id-subst}) \cdot v \varrho = x \cdot v \text{ id-subst}$

*<proof>*

**lemma** *inv-renaming*:

**assumes** *exists-nonground*: *exists-nonground* **and**  $\varrho$ : *is-renaming*  $\varrho$

**shows**  $\text{inv } (\text{var-subst } \varrho) (x \cdot v \varrho) = x$

*<proof>*

**lemma** *renaming-inv-in-vars*:

**assumes**  $\varrho$ : *is-renaming*  $\varrho$  **and**  $x: x \in \text{vars } (expr \cdot \varrho)$

**shows**  $\text{inv } (\text{var-subst } \varrho) (x \cdot v \text{ id-subst}) \in \text{vars } expr$

*<proof>*

**lemma** *inj-id-subst*:

**assumes** *exists-nonground*

**shows** *inj*  $(\text{var-subst } id\text{-subst})$

*<proof>*

**end**

**locale** *grounding = substitution* **where**  $\text{vars} = \text{vars}$  **and**  $\text{id-subst} = \text{id-subst}$

**for**  $\text{vars} :: 'expr \Rightarrow 'var \text{ set}$  **and**  $\text{id-subst} :: 'subst +$

**fixes**  $\text{to-ground} :: 'expr \Rightarrow 'expr_G$  **and**  $\text{from-ground} :: 'expr_G \Rightarrow 'expr$

**assumes**

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

**and**

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

**begin**

**definition** *ground-instances'*  $:: 'expr \Rightarrow 'expr_G \text{ set}$  **where**

$\text{ground-instances}' \text{ expr} = \{ \text{to-ground } (expr \cdot \gamma) \mid \gamma. \text{is-ground } (expr \cdot \gamma) \}$

**lemma** *ground-instances'-eq-ground-instances*:  
*ground-instances' expr = (to-ground ' ground-instances expr)*  
 ⟨*proof*⟩

**lemma** *to-ground-from-ground-id* [*simp*]: *to-ground*  $\circ$  *from-ground* = *id*  
 ⟨*proof*⟩

**lemma** *surj-to-ground*: *surj to-ground*  
 ⟨*proof*⟩

**lemma** *inj-from-ground*: *inj-on from-ground domain<sub>G</sub>*  
 ⟨*proof*⟩

**lemma** *inj-on-to-ground*: *inj-on to-ground (from-ground ' domain<sub>G</sub>)*  
 ⟨*proof*⟩

**lemma** *bij-betw-to-ground*: *bij-betw to-ground (from-ground ' domain<sub>G</sub>) domain<sub>G</sub>*  
 ⟨*proof*⟩

**lemma** *bij-betw-from-ground*: *bij-betw from-ground domain<sub>G</sub> (from-ground ' domain<sub>G</sub>)*  
 ⟨*proof*⟩

**lemma** *ground-is-ground* [*simp, intro*]: *is-ground (from-ground expr<sub>G</sub>)*  
 ⟨*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<sub>G</sub> where from-ground expr<sub>G</sub> = 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** *exists-ground* = *substitution* +  
**assumes** *exists-ground*:  $\exists \text{expr}. \text{is-ground } \text{expr}$

**locale** *exists-ground-subst* = *substitution* +  
**assumes** *exists-ground-subst*:  $\exists \gamma. \text{is-ground-subst } \gamma$   
**begin**

**lemma** *obtain-ground-subst*:  
**obtains**  $\gamma$  **where** *is-ground-subst*  $\gamma$   
 $\langle \text{proof} \rangle$

**sublocale** *exists-ground*  
 $\langle \text{proof} \rangle$

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

**end**

**locale** *exists-imgu* = *substitution* +  
**assumes**  
 $\bigwedge v \text{ expr expr}'. \text{exists-nonground} \implies \text{expr} \cdot v = \text{expr}' \cdot v \implies \exists \mu. \text{is-imgu } \mu$   
 $\{\{\text{expr}, \text{expr}'\}\}$   
**begin**

**lemma** *exists-imgu*:  
**assumes**  $\text{expr} \cdot v = \text{expr}' \cdot v$   
**shows**  $\exists \mu. \text{is-imgu } \mu \{\{\text{expr}, \text{expr}'\}\}$   
 $\langle \text{proof} \rangle$

**lemma** *obtains-imgu*:  
**assumes**  $\text{expr} \cdot v = \text{expr}' \cdot v$   
**obtains**  $\mu$  **where** *is-imgu*  $\mu \{\{\text{expr}, \text{expr}'\}\}$   
 $\langle \text{proof} \rangle$

**lemma** *exists-imgu-set*:  
**assumes**  
*finite-X*: *finite*  $X$  **and**  
*unifier*: *is-unifier*  $v X$   
**shows**  $\exists \mu. \text{is-imgu } \mu \{X\}$   
 $\langle \text{proof} \rangle$

**lemma** *exists-imgu-sets*:  
**assumes**

```

    finite-XX: finite XX and
    finite-X:  $\forall X \in XX. \textit{finite } X$  and
    unifier: is-unifier-set v XX
    shows  $\exists \mu. \textit{is-imgu } \mu \textit{ } XX$ 
    <proof>

```

**end**

```

locale subst-updates-compat =
  subst-update +
  assumes subst-updates-compat:
     $\bigwedge \textit{expr } \sigma. \forall x \in \textit{vars expr}. \textit{fmlookup } u \ x = \textit{Some } (x \cdot v \ \sigma) \implies \textit{expr} \cdot \textit{id-subst}[[u]]$ 
    = expr  $\cdot$   $\sigma$ 

```

```

locale subst-eq =
  substitution +
  assumes subst-eq:  $\bigwedge \textit{expr } \sigma \ \tau. (\bigwedge x. x \in \textit{vars expr} \implies x \cdot v \ \sigma = x \cdot v \ \tau) \implies \textit{expr}$ 
     $\cdot$   $\sigma = \textit{expr} \cdot \tau$ 
begin

```

```

lemma subset-subst-eq:
  assumes  $\forall x \in \textit{vars } C. x \cdot v \ \sigma_1 = x \cdot v \ \sigma_2 \ \textit{vars } D \subseteq \textit{vars } C$ 
  shows  $D \cdot \sigma_1 = D \cdot \sigma_2$ 
  <proof>

```

**end**

```

locale is-ground-if-no-vars = substitution +
  assumes is-ground-if-no-vars:  $\bigwedge \textit{expr}. \textit{vars expr} = \{\} \implies \textit{is-ground expr}$ 
begin

```

```

lemma is-ground-iff-no-vars: is-ground expr  $\longleftrightarrow \textit{vars expr} = \{\}$ 
  <proof>

```

**end**

```

end
theory Based-Substitution
  imports Substitution
begin

```

## 9 Substitutions on base expressions

```

locale base-substitution = substitution where vars = vars and apply-subst = apply-subst
  for vars :: 'base  $\Rightarrow$  'v set and apply-subst :: 'v  $\Rightarrow$  'subst  $\Rightarrow$  'base (infixl  $\langle \cdot v \rangle$  69)
  +
  assumes

```

— The precondition of the assumption ensures noop substitutions  
*vars-id-subst*:  $\bigwedge x. \text{exists-nonground} \implies \text{vars } (x \cdot v \text{ id-subst}) = \{x\}$  **and**  
*comp-subst-iff*:  $\bigwedge \sigma \sigma' x. x \cdot v \sigma \odot \sigma' = \text{subst } (x \cdot v \sigma) \sigma'$  **and**  
*base-vars-subst*:  $\bigwedge \text{expr } \sigma. \text{vars } (\text{expr} \cdot \sigma) = \bigcup (\text{vars } \text{'var-subst } \sigma \text{' vars expr})$   
**and**  
*base-vars-grounded-if-is-grounding*:  
 $\bigwedge \text{expr } \gamma. \text{is-ground } (\text{expr} \cdot \gamma) \implies \forall x \in \text{vars expr}. \text{is-ground } (x \cdot v \gamma)$

**locale** *based-substitution* =  
*substitution* **where** *vars* = *vars* **and** *apply-subst* = *apply-subst* :: *'v*  $\Rightarrow$  *'subst*  $\Rightarrow$   
*'base* +  
*base*: *base-substitution* **where**  
*subst* = *base-subst* **and** *vars* = *base-vars* **and** *is-ground* = *base-is-ground*  
**for**  
*base-subst* :: *'base*  $\Rightarrow$  *'subst*  $\Rightarrow$  *'base* **and**  
*base-vars* :: *'base*  $\Rightarrow$  *'v set* **and**  
*vars* :: *'expr*  $\Rightarrow$  *'v set* **and**  
*base-is-ground* +  
**assumes**  
*exists-nonground-iff-base-exists-nonground [simp]*: *exists-nonground*  $\longleftrightarrow$  *base.exists-nonground*  
**and**  
*vars-subst*:  $\bigwedge \text{expr } \varrho. \text{vars } (\text{expr} \cdot \varrho) = \bigcup (\text{base-vars } \text{'var-subst } \varrho \text{' vars expr})$   
**and**  
*vars-grounded-if-is-grounding*:  
 $\bigwedge \text{expr } \gamma. \text{is-ground } (\text{expr} \cdot \gamma) \implies \forall x \in \text{vars expr}. \text{base-is-ground } (x \cdot v \gamma)$   
**begin**

**lemma** *id-subst-subst [simp]*: *base-subst*  $(x \cdot v \text{ id-subst}) \sigma = x \cdot v \sigma$   
*<proof>*

**lemma** *variable-grounding*:  
**assumes** *is-ground*  $(\text{expr} \cdot \gamma)$   $x \in \text{vars expr}$   
**shows** *base-is-ground*  $(x \cdot v \gamma)$   
*<proof>*

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

**lemma** *vars-id-subst-subset*: *base-vars*  $(x \cdot v \text{ id-subst}) \subseteq \{x\}$   
*<proof>*

**lemma** *vars-subst-subset*: *vars*  $(\text{expr} \cdot \sigma) \subseteq (\text{vars expr} - \text{subst-domain } \sigma) \cup$   
*range-vars*  $\sigma$   
*<proof>*

**end**

**context** *base-substitution*  
**begin**

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

**declare** *exists-nonground-iff-base-exists-nonground* [*simp del*]

**end**

**hide-fact** *base-substitution.base-vars-subst*  
**hide-fact** *base-substitution.base-vars-grounded-if-is-grounding*

## 10 Properties of substitutions on base expressions

**locale** *based-subst-update* =  
*based-substitution* +  
*subst-update* +  
**assumes** *ground-subst-update-in-vars*:  
 $\bigwedge \text{update expr } \gamma \ x. \text{ base-is-ground update} \implies \text{is-ground (expr } \cdot \gamma) \implies x \in \text{vars expr} \implies$   
 $\text{is-ground (expr } \cdot \gamma[x := \text{update}])$   
**begin**

**lemma** *vars-id-subst-update*:  $\text{vars (expr } \cdot \text{id-subst}[x := b]) \subseteq \text{vars expr} \cup \text{base-vars } b$   
 ⟨*proof*⟩

**lemma** *ground-subst-update* [*simp*]:  
**assumes** *base-is-ground update is-ground (expr } \cdot \gamma)*  
**shows** *is-ground (expr } \cdot \gamma[x := update])*  
 ⟨*proof*⟩

**end**

**locale** *create-renaming* = *based-subst-update* **where**  
*apply-subst* = *apply-subst* **for**  
*apply-subst* :: 'v  $\Rightarrow$  'subst  $\Rightarrow$  'base (**infixl**  $\cdot v$  69) +  
**assumes** *id-fold-subst-comp-ext*:  
 $\bigwedge us \ us'. \text{ exists-nonground} \implies$   
 $(\bigwedge x. x \cdot v \text{ fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us \ \text{id-subst} \odot \text{fold } (\lambda(x, b) \sigma. \sigma[x := b])$   
 $us' \ \text{id-subst}$   
 $= x \cdot v \ \text{id-subst}) \implies$   
 $\text{fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us \ \text{id-subst} \odot \text{fold } (\lambda(x, b) \sigma. \sigma[x := b]) \ us' \ \text{id-subst}$   
 $= \text{id-subst}$   
**begin**

**lemma** *create-renaming*:  
**assumes** *exists-nonground: exists-nonground and finite-S: finite S and bij: bij-betw f S T*

**shows** *is-renaming* (*renaming-of-bij*  $f S T$ )  
 ⟨*proof*⟩

**end**

**locale** *variables-in-base-imgu* = *based-substitution* +  
**assumes** *variables-in-base-imgu*:

$\bigwedge expr \mu XX.$   
 $base.is\text{-}imgu \mu XX \implies finite\ XX \implies \forall X \in XX. finite\ X \implies$   
 $vars\ (expr \cdot \mu) \subseteq vars\ expr \cup (\bigcup (base\text{-}vars\ ' \bigcup XX))$

**locale** *range-vars-subset-if-is-imgu* = *base-substitution* +  
**assumes** *range-vars-subset-if-is-imgu*:

$\bigwedge \mu XX. is\text{-}imgu \mu XX \implies \forall X \in XX. finite\ X \implies finite\ XX \implies$   
 $range\text{-}vars\ \mu \subseteq (\bigcup expr \in \bigcup XX. vars\ expr)$

**begin**

**sublocale** *variables-in-base-imgu* **where**

*base\text{-}subst* = *subst* **and** *base\text{-}vars* = *vars* **and** *base\text{-}is\text{-}ground* = *is\text{-}ground*  
 ⟨*proof*⟩

**end**

**locale** *base-variables-in-base-imgu* =

*base\text{-}substitution* **where** *vars* = *vars* :: '*expr*  $\Rightarrow$  '*v* *set* +  
*subst\text{-}update* +  
*finite\text{-}variables* +  
*infinite\text{-}variables* +

**assumes**

*not\text{-}back\text{-}to\text{-}id\text{-}subst*:  $\bigwedge expr\ \sigma. \exists x. expr \cdot \sigma = x \cdot v\ id\text{-}subst \implies \exists x. expr = x \cdot v$   
*id\text{-}subst*

**begin**

**lemma** *imgu\text{-}subst\text{-}domain\text{-}subset*:

**assumes** *imgu*: *is\text{-}imgu*  $\mu XX$   
**shows** *subst\text{-}domain*  $\mu \subseteq \bigcup (vars\ ' \bigcup XX)$

⟨*proof*⟩

**lemma** *imgu\text{-}subst\text{-}domain\text{-}finite*:

**assumes** *imgu*: *is\text{-}imgu*  $\mu XX$  **and** *finite\text{-}X*:  $\forall X \in XX. finite\ X$  **and** *finite\text{-}XX*:  
*finite XX*

**shows** *finite* (*subst\text{-}domain*  $\mu$ )

⟨*proof*⟩

**lemma** *imgu\text{-}range\text{-}vars\text{-}finite*:

**assumes** *imgu*: *is\text{-}imgu*  $\mu XX$  **and** *finite\text{-}X*:  $\forall X \in XX. finite\ X$  **and** *finite\text{-}XX*:  
*finite XX*

**shows** *finite* (*range\text{-}vars*  $\mu$ )

*<proof>*

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

**assumes** *imgu*: *is-imgu*  $\mu$  *XX* **and** *finite-X*:  $\forall X \in XX. \text{finite } X$  **and** *finite-XX*:  
*finite* *XX*

**shows**  $\bigcup (\text{vars } ' \text{expr-subst } \mu ' \bigcup XX) \subseteq \bigcup (\text{vars } ' \bigcup XX)$   
*<proof>*

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

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

**shows** *range-vars*  $\mu \subseteq (\bigcup \text{expr} \in \bigcup XX. \text{vars } \text{expr})$   
*<proof>*

**sublocale** *variables-in-base-imgu* **where**

*base-subst* = *subst* **and** *base-vars* = *vars* **and** *base-is-ground* = *is-ground*

*<proof>*

**end**

**locale** *base-exists-non-ident-subst* =

*base-substitution* **where** *vars* = *vars* +  
*finite-variables* **where** *vars* = *vars* +  
*infinite-variables* **where** *vars* = *vars* +  
*all-subst-ident-iff-ground* **where** *vars* = *vars* +  
*subst-update* **where** *vars* = *vars* +  
*is-ground-if-no-vars* **where** *vars* = *vars*  
**for** *vars* :: 'expr  $\Rightarrow$  'v set

**begin**

**sublocale** *exists-non-ident-subst*

*<proof>*

**end**

**locale** *vars-grounded-iff-is-grounding* = *based-substitution* +

**assumes** *is-grounding-if-vars-grounded*:

$\bigwedge \text{expr } \gamma. \forall x \in \text{vars } \text{expr}. \text{base-is-ground } (x \cdot v \ \gamma) \implies \text{is-ground } (\text{expr} \cdot \gamma)$

**begin**

**lemma** *vars-grounded-iff-is-grounding*:  $(\forall x \in \text{vars } b. \text{base-is-ground } (x \cdot v \ \gamma)) \iff$   
*is-ground*  $(b \cdot \gamma)$

*<proof>*

**end**

**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  $\bigwedge a b. \text{add } a \ b \equiv \text{plus } (\text{wrap } a) \ b$ 
  assumes
    to-set-plus [simp]:  $\bigwedge b \ b'. \text{to-set } (\text{plus } b \ b') = (\text{to-set } b) \cup (\text{to-set } b')$  and
    to-set-wrap [simp]:  $\bigwedge a. \text{to-set } (\text{wrap } a) = \{a\}$ 
begin

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

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  $\Rightarrow$  'a)  $\Rightarrow$  'b  $\Rightarrow$  'b and
    to-set :: 'b  $\Rightarrow$  'a set
  assumes
    map-comp [simp]:  $\bigwedge b \ f \ g. \text{map } f \ (\text{map } g \ b) = \text{map } (f \ o \ g) \ b$  and
    map-ident [simp]:  $\bigwedge b. \text{map } (\lambda x. x) \ b = b$  and
    map-cong0 [cong]:  $\bigwedge b \ f \ g. (\bigwedge a. a \in \text{to-set } b \Longrightarrow f \ a = g \ a) \Longrightarrow \text{map } f \ b = \text{map}$ 
     $g \ b$  and
    to-set-map [simp]:  $\bigwedge b \ f. \text{to-set } (\text{map } f \ b) = f \ ` \ \text{to-set } b$ 
begin

lemma map-comp' [simp]:  $\bigwedge b \ f \ g. \text{map } f \ (\text{map } g \ b) = \text{map } (\lambda x. f \ (g \ x)) \ b$ 
   $\langle \text{proof} \rangle$ 

lemma map-id [simp]:  $\text{map } \text{id} \ b = b$ 
   $\langle \text{proof} \rangle$ 

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

```

**shows**  $\text{map } f \ b = \text{map } g \ b'$   
 ⟨proof⟩

**lemma** *map-id-cong* [*simp*]:  
**assumes**  $\bigwedge a. a \in \text{to-set } b \implies f \ a = a$   
**shows**  $\text{map } f \ b = b$   
 ⟨proof⟩

**lemma** *to-set-map-not-ident*:  
**assumes**  $a \in \text{to-set } b \ f \ a \notin \text{to-set } b$   
**shows**  $\text{map } f \ b \neq b$   
 ⟨proof⟩

**lemma** *map-in-to-set*:  
**assumes**  $\text{map } f \ b = b \ a \in \text{to-set } b$   
**shows**  $f \ a \in \text{to-set } b$   
 ⟨proof⟩

**lemma** *to-set-const* [*simp*]:  $\text{to-set } b \neq \{\}$   $\implies \text{to-set } (\text{map } (\lambda-. a) \ b) = \{a\}$   
 ⟨proof⟩

**lemma** *map-inverse*:  $(\bigwedge x. f \ (g \ x) = x) \implies \text{map } f \ (\text{map } g \ b) = b$   
 ⟨proof⟩

**end**

**locale** *non-empty-natural-functor* = *natural-functor* +  
**assumes** *exists-non-empty*:  $\exists b. \text{to-set } b \neq \{\}$   
**begin**

**lemma** *exists-functor* [*intro*]:  $\exists b. a \in \text{to-set } b$   
 ⟨proof⟩

**end**

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

**locale** *non-empty-finite-natural-functor* =  
*non-empty-natural-functor* + *finite-natural-functor*

**locale** *natural-functor-conversion* =  
*natural-functor* +  
*functor'*: *natural-functor* **where**  $\text{map} = \text{map}'$  **and**  $\text{to-set} = \text{to-set}'$   
**for**  $\text{map}' :: ('b \Rightarrow 'b) \Rightarrow 'd \Rightarrow 'd$  **and**  $\text{to-set}' :: 'd \Rightarrow 'b \ \text{set} +$   
**fixes**  
 $\text{map-to} :: ('a \Rightarrow 'b) \Rightarrow 'c \Rightarrow 'd$  **and**  
 $\text{map-from} :: ('b \Rightarrow 'a) \Rightarrow 'd \Rightarrow 'c$

**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]:  $\bigwedge d f g. \text{map-to } f (\text{map-from } g d) = \text{map}' (\lambda x. f (g x)) d$

**lemma** *non-empty-helper*:  $x \in \text{to-set } b \implies \exists b. \text{to-set } b \neq \{\}$   
 ⟨proof⟩

⟨ML⟩

**end**  
**theory** *Natural-Magma-Function*  
**imports** *Natural-Magma Natural-Function*  
**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** *Substitution-Lifting*  
**imports** *Substitution Natural-Magma-Function*  
**begin**

## 11 Lifting of substitutions using natural functors

**locale** *substitution-lifting* =  
*sub*: *substitution* **where**  
*subst* = *sub-subst* **and** *vars* = *sub-vars* **and** *apply-subst* = *apply-subst* **and**  
*is-ground* = *sub-is-ground* +  
*non-empty-natural-functor* **where** *map* = *map* **and** *to-set* = *to-set*  
**for**  
*sub-vars* :: '*sub*  $\Rightarrow$  '*v* set **and**  
*sub-subst* :: '*sub*  $\Rightarrow$  '*subst*  $\Rightarrow$  '*sub* **and**  
*sub-is-ground* :: '*sub*  $\Rightarrow$  bool **and**  
*map* :: ('*sub*  $\Rightarrow$  '*sub*)  $\Rightarrow$  '*expr*  $\Rightarrow$  '*expr* **and**  
*to-set* :: '*expr*  $\Rightarrow$  '*sub* set **and**  
*apply-subst* :: '*v*  $\Rightarrow$  '*subst*  $\Rightarrow$  '*base* (infixl ·v 69)

**begin**

**definition**  $vars :: 'expr \Rightarrow 'v\ set$  **where**  
 $vars\ expr \equiv \bigcup (sub\ vars\ 'to\ set\ expr)$

**definition**  $is\ ground :: 'expr \Rightarrow bool$  **where**  
 $is\ ground\ expr \equiv \forall sub \in to\ set\ expr. sub\ is\ ground\ sub$

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

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

**lemma**  $subst\ in\ to\ set\ subst$  [intro]:  
**assumes**  $sub \in to\ set\ expr$   
**shows**  $sub \cdot_s \sigma \in to\ set\ (expr \cdot \sigma)$   
(proof)

**sublocale**  $substitution$  **where**  $subst = (\cdot)$  **and**  $vars = vars$  **and**  $is\ ground = is\ ground$   
(proof)

**lemma**  $exists\ nonground\ iff\ sub\ exists\ nonground$ :  $exists\ nonground \longleftrightarrow sub.exists\ nonground$   
(proof)

**lemma**  $ground\ subst\ iff\ sub\ ground\ subst$  [simp]:  $is\ ground\ subst\ \gamma \longleftrightarrow sub.is\ ground\ subst\ \gamma$   
(proof)

**lemma**  $to\ set\ is\ ground$  [intro]:  
**assumes**  $sub \in to\ set\ expr$   $is\ ground\ expr$   
**shows**  $sub\ is\ ground\ sub$   
(proof)

**lemma**  $to\ set\ is\ ground\ subst$ :  
**assumes**  $sub \in to\ set\ expr$   $is\ ground\ (expr \cdot \gamma)$   
**shows**  $sub\ is\ ground\ (sub \cdot_s \gamma)$   
(proof)

**lemma**  $subst\ empty$ :  
**assumes**  $to\ set\ expr' = \{\}$   
**shows**  $expr \cdot \sigma = expr' \longleftrightarrow expr = expr'$   
(proof)

**lemma**  $empty\ is\ ground$ :  
**assumes**  $to\ set\ expr = \{\}$   
**shows**  $is\ ground\ expr$   
(proof)

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

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

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

**end**

## 11.1 Lifting of properties

**locale** *subst-update-lifting* =  
*sub*: *subst-update* **where**  
 $vars = sub\text{-vars} :: 'sub \Rightarrow 'v \text{ set}$  **and**  $subst = sub\text{-subst}$  **and**  $is\text{-ground} = sub\text{-is-ground}$   
 +  
*substitution-lifting*  
**begin**

**sublocale** *subst-update* **where**  $vars = vars$  **and**  $subst = subst$  **and**  $is\text{-ground} = is\text{-ground}$   
 ⟨*proof*⟩

**end**

**locale** *finite-variables-lifting* =  
*sub*: *finite-variables* **where**  
 $vars = sub\text{-vars} :: 'sub \Rightarrow 'v \text{ set}$  **and**  $subst = sub\text{-subst}$  **and**  $is\text{-ground} = sub\text{-is-ground}$   
 +

*finite-natural-functor* **where**  $to\text{-set} = to\text{-set} :: 'expr \Rightarrow 'sub \text{ set}$  +

*substitution-lifting*

**begin**

**abbreviation**  $to\text{-fset} :: 'expr \Rightarrow 'sub \text{ fset}$  **where**  
 $to\text{-fset } expr \equiv Abs\text{-fset} (to\text{-set } expr)$

**sublocale** *finite-variables* **where**  $vars = vars$  **and**  $subst = subst$  **and**  $is\text{-ground} = is\text{-ground}$   
 ⟨*proof*⟩

**lemma** *fset-to-fset* [*simp*]:  $fset (to\text{-fset } expr) = to\text{-set } expr$   
 ⟨*proof*⟩

**lemma** *to-fset-map*:  $to-fset (map f expr) = f \mid\mid to-fset expr$   
*<proof>*

**lemma** *to-fset-is-ground-subst*:  
**assumes**  $sub \mid\in\mid to-fset expr is-ground (subst expr \gamma)$   
**shows**  $sub-is-ground (sub \cdot_s \gamma)$   
*<proof>*

**end**

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

**sublocale** *renaming-variables* **where**  $subst = subst$  **and**  $vars = vars$  **and**  $is-ground = is-ground$   
*<proof>*

**end**

**locale** *exists-ground-lifting* =  
  *substitution-lifting* +  
  *sub*: *exists-ground* **where**  $vars = sub-vars$  **and**  $subst = sub-subst$  **and**  $is-ground = sub-is-ground$   
**begin**

**sublocale** *exists-ground* **where**  $vars = vars$  **and**  $subst = subst$  **and**  $is-ground = is-ground$   
*<proof>*

**end**

**locale** *grounding-lifting* =  
  *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**  
   $from-ground = sub-from-ground$  **and**  $is-ground = sub-is-ground$  +

*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 :: 'subG ⇒ 'sub and
sub-vars :: 'sub ⇒ 'var set and
sub-subst :: 'sub ⇒ 'subst ⇒ 'sub and
map :: ('sub ⇒ 'sub) ⇒ 'expr ⇒ 'expr and
to-ground-map :: ('sub ⇒ 'subG) ⇒ 'expr ⇒ 'exprG and
from-ground-map :: ('subG ⇒ 'sub) ⇒ 'exprG ⇒ 'expr and
ground-map :: ('subG ⇒ 'subG) ⇒ 'exprG ⇒ 'exprG and
to-set-ground :: 'exprG ⇒ 'subG set
begin

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

definition from-ground :: 'exprG ⇒ 'expr where
  from-ground exprG ≡ from-ground-map sub-from-ground exprG

sublocale grounding where
  vars = vars and subst = subst and to-ground = to-ground and from-ground =
  from-ground and
  is-ground = is-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:
  (∀ sub ∈ to-set (from-ground exprG). P sub) ⟷
  (∀ subG ∈ to-set-ground exprG. P (sub-from-ground subG))
  ⟨proof⟩

end

locale exists-non-ident-subst-lifting =
  finite-variables-lifting where map = map +
  sub: exists-non-ident-subst where subst = sub-subst and vars = sub-vars and
  is-ground = sub-is-ground
for map :: ('sub ⇒ 'sub) ⇒ 'expr ⇒ 'expr
begin

```

**sublocale** *exists-non-ident-subst* **where**  $subst = subst$  **and**  $vars = vars$  **and**  $is-ground = is-ground$   
 $\langle proof \rangle$

**end**

**locale** *all-subst-ident-iff-ground-lifting* =  
*exists-non-ident-subst-lifting* **where**  $map = map +$   
 $sub: all-subst-ident-iff-ground$  **where**  
 $subst = sub-subst$  **and**  $vars = sub-vars$  **and**  $is-ground = sub-is-ground$   
**for**  $map :: ('sub \Rightarrow 'sub) \Rightarrow 'expr \Rightarrow 'expr$   
**begin**

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

**end**

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

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

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

**lemma** *is-ground-plus* [*simp*]:  
 $is-ground (plus\ expr\ expr') \iff is-ground\ expr \wedge is-ground\ expr'$   
 $\langle proof \rangle$

**lemma** *is-ground-add* [*simp*]:  
 $is-ground (add\ sub\ expr) \iff sub-is-ground\ sub \wedge is-ground\ expr$   
 $\langle proof \rangle$

**end**

**locale** *natural-magma-functor-substitution-lifting* =  
*natural-magma-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*]:  $(plus\ expr\ expr') \cdot \sigma = plus (expr \cdot \sigma) (expr' \cdot \sigma)$

$\langle \text{proof} \rangle$

**end**

**locale** *natural-magma-grounding-lifting* =  
*grounding-lifting* +  
*natural-magma-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*]:  
 $\bigwedge \text{expr expr}'. \text{to-ground } (\text{plus expr expr}') = \text{plus-ground } (\text{to-ground expr}) (\text{to-ground expr}')$  **and**  
*wrap-from-ground*:  $\bigwedge \text{sub}. \text{from-ground } (\text{wrap-ground sub}) = \text{wrap } (\text{sub-from-ground sub})$  **and**  
*wrap-to-ground*:  $\bigwedge \text{sub}. \text{to-ground } (\text{wrap sub}) = \text{wrap-ground } (\text{sub-to-ground sub})$   
**begin**

**lemma** *from-ground-plus* [*simp*]:  
 $\text{from-ground } (\text{plus-ground expr expr}') = \text{plus } (\text{from-ground expr}) (\text{from-ground expr}')$   
 $\langle \text{proof} \rangle$

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

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

**lemma** *ground-add*:  
**assumes** *from-ground* *expr* = *add sub expr'*  
**shows** *expr* = *add-ground* (*sub-to-ground sub*) (*to-ground expr'*)  
 $\langle \text{proof} \rangle$

**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

locale exists-ground-subst-lifting =
  substitution-lifting +
  sub: exists-ground-subst where vars = sub-vars and subst = sub-subst and
is-ground = sub-is-ground
begin

sublocale exists-ground-subst where vars = vars and subst = subst and is-ground
  = is-ground
  ⟨proof⟩

end

locale subst-updates-compat-lifting =
  subst-update-lifting +
  sub: subst-updates-compat where
  vars = sub-vars and subst = sub-subst and is-ground = sub-is-ground
begin

sublocale subst-updates-compat where vars = vars and subst = subst and is-ground
  = is-ground
  ⟨proof⟩

end

locale subst-eq-lifting =
  sub: subst-eq where
  vars = sub-vars :: 'sub  $\Rightarrow$  'v set and subst = sub-subst and is-ground = sub-is-ground

```

```

+
  substitution-lifting
begin

sublocale subst-eq where vars = vars and subst = subst and is-ground = is-ground
⟨proof⟩

end

locale is-ground-if-no-vars-lifting =
  substitution-lifting +
  sub: is-ground-if-no-vars where
    vars = sub-vars and is-ground = sub-is-ground and subst = sub-subst
begin

sublocale is-ground-if-no-vars where vars = vars and is-ground = is-ground and
subst = subst
  ⟨proof⟩

end

end
theory Based-Substitution-Lifting
  imports
    Based-Substitution
    Substitution-Lifting
begin

```

## 12 Lifting of based substitutions

```

locale based-substitution-lifting =
  substitution-lifting +
  base: base-substitution where
    subst = base-subst and vars = base-vars and is-ground = base-is-ground +
    sub: based-substitution where
      subst = sub-subst and vars = sub-vars and is-ground = sub-is-ground
begin

sublocale based-substitution where subst = subst and vars = vars and is-ground
= is-ground
  ⟨proof⟩

end

```

### 12.1 Lifting of properties

```

locale variables-in-base-imgu-lifting =
  based-substitution-lifting +
  sub: variables-in-base-imgu where

```

```

    vars = sub-vars and subst = sub-subst and is-ground = sub-is-ground
begin

sublocale variables-in-base-ingu where subst = subst and vars = vars and
is-ground = is-ground
⟨proof⟩

end

locale based-subst-update-lifting =
  based-substitution-lifting +
  subst-update-lifting +
  sub: based-subst-update where
    vars = sub-vars and subst = sub-subst and is-ground = sub-is-ground
begin

sublocale based-subst-update where subst = subst and vars = vars and is-ground
= is-ground
⟨proof⟩

end

locale vars-grounded-iff-is-grounding-lifting =
  based-substitution-lifting +
  sub: vars-grounded-iff-is-grounding where
    vars = sub-vars and subst = sub-subst and is-ground = sub-is-ground
begin

sublocale vars-grounded-iff-is-grounding where
  subst = subst and vars = vars and is-ground = is-ground
  ⟨proof⟩

end

end
theory Noop-Substitution
  imports Based-Substitution
begin

definition noop-comp-subst where
  noop-comp-subst  $\sigma - = \sigma$ 

definition noop-id-subst where
  noop-id-subst = ()

global-interpretation unit: abstract-substitution-monoid where
  comp-subst = noop-comp-subst and id-subst = noop-id-subst
  ⟨proof⟩

```

**locale** *properties* =  
*base-substitution* +  
*all-subst-ident-iff-ground* +  
*exists-non-ident-subst* +  
*subst-update* +  
*grounding* +  
*finite-variables* +  
*renaming-variables* +  
*exists-ground* +  
*range-vars-subset-if-is-imgu* +  
*exists-imgu* +  
 create-renaming **where** *base-subst* = *subst* **and** *base-vars* = *vars* **and** *base-is-ground*  
 = *is-ground*

**definition** *noop-apply-subst* **where**  
*noop-apply-subst* - -  $\equiv$  *SOME* *expr*. *True*

**definition** *noop-subst* **where**  
*noop-subst* *expr* -  $\equiv$  *expr*

**definition** *noop-vars* **where**  
*noop-vars* -  $\equiv$  {}

**definition** *noop-is-ground* **where**  
*noop-is-ground* -  $\longleftrightarrow$  *True*

**definition** *noop-subst-update* **where**  
*noop-subst-update*  $\sigma$  - -  $\equiv$   $\sigma$

**context** *abstract-substitution-monoid*  
**begin**

**sublocale** *noop: base-substitution* **where**  
*vars* = *noop-vars* **and** *apply-subst* = *noop-apply-subst* **and** *is-ground* = *noop-is-ground*  
**and**  
*subst* = *noop-subst*  
 ⟨*proof*⟩

**sublocale** *noop: properties* **where**  
*vars* = *noop-vars* **and** *subst-update* = *noop-subst-update* **and** *apply-subst* =  
*noop-apply-subst* **and**  
*subst* = *noop-subst* **and** *to-ground* = *id* **and** *from-ground* = *id* **and** *is-ground* =  
*noop-is-ground*  
 ⟨*proof*⟩

**lemma** *noop-subst [simp]*: *noop-subst* *expr*  $\sigma$  = *expr*  
 ⟨*proof*⟩

**lemma** *noop-is-ground* [*simp*]: *noop-is-ground* *expr*  
 ⟨*proof*⟩

**lemma** *noop-vars* [*simp*]: *noop-vars* *expr* = {}  
 ⟨*proof*⟩

**end**

**end**

**theory** *Substitution-First-Order-Term*

**imports**

*Based-Substitution*

*First-Order-Terms.Unification*

*Regular-Tree-Relations.Ground-Terms*

**begin**

## 13 Substitutions for first order terms

### 13.1 Interpretations for first order terms

**abbreviation** (*input*) *apply-subst* **where**  
*apply-subst* *x*  $\sigma \equiv \sigma$  *x*

**abbreviation** (*input*) *is-ground* **where**  
*is-ground* *t*  $\equiv$  *vars-term* *t* = {}

**global-interpretation** *term*: *base-substitution* **where**  
*comp-subst* = ( $\circ_s$ ) **and** *id-subst* = *Var* **and** *subst* = ( $\cdot$ ) **and** *vars* = *vars-term*  
**and**  
*apply-subst* = *apply-subst* **and** *is-ground* = *is-ground*  
 ⟨*proof*⟩

**lemma** *term-is-renaming-iff*: *term.is-renaming*  $\rho \longleftrightarrow$  *inj*  $\rho \wedge (\forall x. \exists x'. \rho x = \text{Var } x')$   
 ⟨*proof*⟩

**locale** *term-properties* =  
*base-substitution* +  
*all-subst-ident-iff-ground* +  
*exists-non-ident-subst* +  
*grounding* +  
*finite-variables* +  
*renaming-variables* +  
*exists-ground* +  
*create-renaming* **where**  
*base-vars* = *vars* **and** *base-subst* = *subst* **and** *base-is-ground* = *is-ground*

**global-interpretation** *term*: *term-properties* **where**

$comp\text{-}subst = (\circ_s)$  **and**  $id\text{-}subst = Var$  **and**  $subst = (\cdot)$  **and**  $subst\text{-}update = fun\text{-}upd$  **and**  
 $apply\text{-}subst = apply\text{-}subst$  **and**  $vars = vars\text{-}term$  **and**  $to\text{-}ground = gterm\text{-}of\text{-}term$   
**and**  
 $from\text{-}ground = term\text{-}of\text{-}gterm$  **and**  $is\text{-}ground = is\text{-}ground$   
 $\langle proof \rangle$

**lemma** *exists-nonground-term* [intro]:  $\exists t. vars\text{-}term\ t \neq \{\}$   
 $\langle proof \rangle$

**lemmas** *term-id-subst-rewrite* = *term.id-subst-rewrite*[OF *exists-nonground-term*]

## 13.2 Compatibility with First\_Order\_Term

Prefer *term.subst-id-subst* to *subst-apply-term-empty*.

**declare** *subst-apply-term-empty*[no-atp]

**lemmas** *term-context-ground-iff-term-is-ground* [simp] = *ground-vars-term-empty*

The other direction does not hold!

**lemma** *Term-is-renaming-if-is-renaming*:

**assumes** *term.is-renaming*  $\varrho$

**shows** *Term.is-renaming*  $\varrho$

$\langle proof \rangle$

**lemma** *Term-subst-domain-eq-subst-domain* [simp]: *Term.subst-domain*  $\sigma = term.subst-domain$   
 $\sigma$

$\langle proof \rangle$

**lemma** *Term-range-vars-eq-range-vars* [simp]: *Term.range-vars*  $\sigma = term.range-vars$   
 $\sigma$

$\langle proof \rangle$

**lemma** *Unifiers-unifiers-Times-iff-is-unifier*:

$\mu \in Unifiers.unifiers\ (X \times X) \longleftrightarrow term.is-unifier\ \mu\ X$

$\langle proof \rangle$

**lemma** *Unifiers-unifiers-Union-Times-iff-is-unifier-set*:

$\mu \in Unifiers.unifiers\ (\bigcup X \in XX. X \times X) \longleftrightarrow term.is-unifier\text{-}set\ \mu\ XX$

$\langle proof \rangle$

**lemma** *Unifiers-is-mgu-Union-Times-iff-is-mgu*:

*Unifiers.is-mgu*  $\mu\ (\bigcup X \in XX. X \times X) \longleftrightarrow term.is-mgu\ \mu\ XX$

$\langle proof \rangle$

**lemma** *Unifiers-is-imgu-Union-Times-iff-is-imgu*:

*Unifiers.is-imgu*  $\mu\ (\bigcup X \in XX. X \times X) \longleftrightarrow term.is-imgu\ \mu\ XX$

$\langle proof \rangle$

**lemma** *Unifiers-is-mgu-iff-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**  $\text{Unifiers.is-imgu } \mu \ X \longleftrightarrow \text{term.is-imgu } \mu \ (\text{set-prod } ' X)$   
*<proof>*

### 13.3 Interpretations for IMGUs

We could also use *base-variables-in-base-imgu*, but would then require infinite variables.

**global-interpretation** *term: range-vars-subset-if-is-imgu* **where**  
 $\text{comp-subst} = (\circ_s)$  **and**  $\text{id-subst} = \text{Var}$  **and**  $\text{subst} = (\cdot)$  **and**  
 $\text{apply-subst} = \text{apply-subst}$  **and**  $\text{vars} = \text{vars-term}$  **and**  $\text{is-ground} = \text{is-ground}$   
*<proof>*

**global-interpretation** *term: exists-imgu* **where**  
 $\text{comp-subst} = (\circ_s)$  **and**  $\text{id-subst} = \text{Var}$  **and**  $\text{subst} = (\cdot)$  **and**  
 $\text{apply-subst} = \text{apply-subst}$  **and**  $\text{vars} = \text{vars-term}$  **and**  $\text{is-ground} = \text{is-ground}$   
*<proof>*

### 13.4 Additional lemmas

**lemmas** *infinite-terms* [*intro*] = *term.infinite-exprs*[*OF exists-nonground-term*]

**lemma** *is-renaming-iff-ex-inj-fun-on-vars*:  $\text{term.is-renaming } \varrho \longleftrightarrow (\exists f. \text{inj } f \wedge \varrho = \text{Var} \circ f)$   
*<proof>*

**end**

**theory** *Substitution-HOL-ex-Unification*

**imports**

*Based-Substitution*

*HOL-ex.Unification*

*Fresh-Identifiers.Fresh*

**begin**

## 14 Substitutions for first order terms as binary tree

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

**quotient-type**  $'v \text{ subst} = ('v \times 'v \text{ trm}) \text{ list} / (\doteq)$   
*<proof>*

### 14.1 Substitution monoid

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

**lift-definition** *subst-id* :: 'v subst  
is [] <proof>

**global-interpretation** *subst-comp*: monoid *subst-comp* *subst-id*  
<proof>

## 14.2 Transfer definitions and lemmas from HOL-ex.Unification

**lift-definition** *subst-apply* :: 'v trm  $\Rightarrow$  'v subst  $\Rightarrow$  'v trm (infixl <·> 61)  
is *Unification.subst*  
<proof>

**lift-definition** *subst-domain* :: 'v subst  $\Rightarrow$  'v set  
is *Unification.subst-domain*  
<proof>

**lift-definition** *range-vars* :: 'v subst  $\Rightarrow$  'v set  
is *Unification.range-vars*  
<proof>

**lift-definition** *Unifier* :: 'v subst  $\Rightarrow$  'v trm  $\Rightarrow$  'v trm  $\Rightarrow$  bool  
is *Unification.Unifier*  
<proof>

**lift-definition** *IMGU* :: 'v subst  $\Rightarrow$  'v trm  $\Rightarrow$  'v trm  $\Rightarrow$  bool  
is *Unification.IMGU*  
<proof>

**lift-definition** *unify* :: 'v trm  $\Rightarrow$  'v trm  $\Rightarrow$  'v subst option  
is *Unification.unify* <proof>

**lemma** *unify-eq-Some-if-Unifier*:  
assumes *Unifier*  $\sigma$   $t$   $t'$   
shows  $\exists \tau. \text{unify } t \ t' = \text{Some } \tau$   
<proof>

**lemma** *unify-computes-IMGU*:  
assumes *unify*  $t$   $t' = \text{Some } \sigma$   
shows *IMGU*  $\sigma$   $t$   $t'$   
<proof>

**lift-definition** *subst-update* :: 'v  $\times$  'v trm  $\Rightarrow$  'v subst  $\Rightarrow$  'v subst  
is (#)  
<proof>

**abbreviation** (*input*) *is-ground* where  
*is-ground*  $t \equiv \text{vars-of } t = \{\}$

### 14.3 Base Substitution

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

*comp-subst = subst-comp and id-subst = subst-id and subst = subst-apply and vars = vars-of and*

*apply-subst =  $\lambda x \sigma. \text{subst-apply (Var } x) \sigma$  and is-ground = is-ground*  
*\langle proof \rangle*

### 14.4 Substitution Properties

**global-interpretation** *term: create-renaming where*

*comp-subst = subst-comp and id-subst = subst-id and subst = subst-apply and vars = vars-of and*

*is-ground = is-ground and apply-subst =  $\lambda x \sigma. \text{subst-apply (Var } x) \sigma$  and subst-update =  $\lambda \sigma x \text{ update. subst-update (x, update) } \sigma$  and base-subst = subst-apply and base-vars = vars-of and base-is-ground = is-ground*  
*\langle proof \rangle*

**global-interpretation** *term: finite-variables where*

*comp-subst = subst-comp and id-subst = subst-id and subst = subst-apply and vars = vars-of and*

*apply-subst =  $\lambda x \sigma. \text{subst-apply (Var } x) \sigma$  and is-ground = is-ground*  
*\langle proof \rangle*

We could also prove *all-subst-ident-iff-ground* and *exists-non-ident-subst* directly without needing infinite variables.

**global-interpretation** *term: base-exists-non-ident-subst where*

*comp-subst = subst-comp and id-subst = subst-id :: 'v :: infinite subst and subst = subst-apply and vars = vars-of and is-ground = is-ground and*

*subst-update =  $\lambda \sigma x \text{ update. subst-update (x, update) } \sigma$  and apply-subst =  $\lambda x \sigma. \text{subst-apply (Var } x) \sigma$*   
*\langle proof \rangle*

**global-interpretation** *term: renaming-variables where*

*comp-subst = subst-comp and id-subst = subst-id :: 'v subst and subst = subst-apply and vars = vars-of and is-ground = is-ground and*

*apply-subst =  $\lambda x \sigma. \text{subst-apply (Var } x) \sigma$*   
*\langle proof \rangle*

### 14.5 Compatibility with HOL-ex.Unification

**lemma** *subst-domain-eq-term-subst-domain [simp]: subst-domain  $\sigma = \text{term.subst-domain } \sigma$*

*\langle proof \rangle*

**lemma** *range-vars-eq-term-range-vars [simp]: range-vars  $\sigma = \text{term.range-vars } \sigma$*

*\langle proof \rangle*

**lemma** *Unifier-iff-term-is-unifier:*

*Unifier  $\mu t t' \longleftrightarrow \text{term.is-unifier } \mu \{t, t'\}$*

*<proof>*

**lemma** *Unifier-iff-is-unifier-set:*

*Unifier*  $\mu$   $t$   $t' \longleftrightarrow \text{term.is-unifier-set } \mu \{\{t, t'\}\}$

*<proof>*

**lemma** *IMGU-iff-term-is-mgu:*

*IMGU*  $\mu$   $t$   $t' \longleftrightarrow \text{term.is-imgu } \mu \{\{t, t'\}\}$

*<proof>*

**lemma** *IMGU-range-vars-subset:*

**assumes** *IMGU*  $\mu$   $t$   $u$

**shows** *range-vars*  $\mu \subseteq \text{vars-of } t \cup \text{vars-of } u$

*<proof>*

## 14.6 Interpretations for IMGUs

We could also prove *range-vars-subset-if-is-imgu* without needing infinite variables.

**global-interpretation** *term: base-variables-in-base-imgu* **where**

*comp-subst* = *subst-comp* **and** *id-subst* = *subst-id* :: '*v* :: *infinite subst* **and**

*subst* = *subst-apply* **and** *vars* = *vars-of* **and** *is-ground* = *is-ground* **and**

*subst-update* =  $\lambda \sigma x \text{ update. subst-update } (x, \text{update}) \sigma$  **and**

*apply-subst* =  $\lambda x \sigma. \text{subst-apply } (\text{Var } x) \sigma$

*<proof>*

**global-interpretation** *term: exists-imgu* **where**

*comp-subst* = *subst-comp* **and** *id-subst* = *subst-id* **and** *subst* = *subst-apply* **and**  
*vars* = *vars-of* **and**

*is-ground* = *is-ground* **and** *apply-subst* =  $\lambda x \sigma. \text{subst-apply } (\text{Var } x) \sigma$

*<proof>*

**end**

**theory** *Substitution-Lifting-Example*

**imports**

*Based-Substitution-Lifting*

*Substitution-First-Order-Term*

**begin**

*<ML>*

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-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* **and** *apply-subst =  $\lambda x \sigma. \sigma x$*  **and**  
*base-is-ground = is-ground* +

*finite-variables-lifting* **where**

*id-subst = Var* **and** *comp-subst = subst-compose* **and** *apply-subst =  $\lambda x \sigma. \sigma x$*

**global-interpretation** *equation-subst:*

*lifting-term-subst-properties* **where**

*sub-vars = vars-term* **and** *sub-subst = subst-apply-term* **and** *sub-is-ground = is-ground* **and**

*map =  $\lambda f. \text{map-prod } f f$*  **and** *to-set = set-prod*

*<proof>*

Lifted lemmas and defintions

**thm**

*equation-subst.vars-id-subst-subset*

*equation-subst.vars-subst-subset*

*equation-subst.to-fset-is-ground-subst*

*equation-subst.vars-def*

*equation-subst.subst-def*

*equation-subst.is-ground-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**

*sub-is-ground = equation-subst.is-ground* **and** *map = fimage* **and** *to-set = fset*

*<proof>*

Lifted lemmas and defintions

**thm**

*equation-set-subst.vars-id-subst-subset*

*equation-set-subst.vars-subst-subset*

*equation-set-subst.to-fset-is-ground-subst*

*equation-set-subst.vars-def*

*equation-set-subst.subst-def*

*equation-set-subst.is-ground-def*

**end**