

First-Order Rewriting

TODO

December 20, 2024

Contents

1	Preliminaries	1
1.1	Combinators	3
1.2	Distinct Lists and Partitions	4
2	Closure-Operations on Relations	5
3	Term Rewriting	7
4	Term Rewrite Systems	9
4.1	Variables of Rules	9
4.2	Variables of TRSs	9
4.3	Function Symbols of Rules	9
4.4	Function Symbols of TRSs	9
4.5	Rewrite steps at a fixed position	37
4.6	Parallel rewrite relation	39
4.7	Function Symbols and Variables	40
4.8	Function Symbols and Variables	40
5	The Critical Pair Lemma	46
6	Multihole Contexts	50
6.1	Partitioning lists into chunks of given length	50
6.2	Conversions from and to multihole contexts	53
6.3	Semilattice Structures	69
6.4	All positions of a multi-hole context	78
6.5	More operations on multi-hole contexts	79
6.6	An inverse of <i>fill-holes</i>	83
6.7	Ditto for <i>fill-holes-mctxt</i>	84
6.8	Function symbols of prefixes	85
6.9	Parallel Rewriting using Multihole Contexts	85
7	Multi-Step Rewriting	87

8	Variables and Variable Positions	93
8.1	Useful abstractions	94
8.2	Additional Functions on Terms	96
8.3	Output	106
8.4	Implementation of parallel rewriting with variable restriction	115
8.5	Implementation of Parallel Rewriting	116
8.6	Generate all root-rewrites (for well-formed TRS)	116
8.7	Generate all parallel rewrites (for well formed TRS)	116
8.8	Implementation of Multistep Rewriting	117
9	A Concrete Unification Algorithm	120
10	Unification of linear and variable disjoint terms	122
11	Sets of Unifiers	126

1 Preliminaries

Some auxiliary results previously located in Auxx.Util of IsaFoR.

theory *Preliminaries*

imports

Polynomial-Factorization.Missing-List

begin

lemma *in-set-idx*: $a \in \text{set } as \implies \exists i. i < \text{length } as \wedge a = as ! i$
 $\langle \text{proof} \rangle$

lemma *finite-card-eq-imp-bij-betw*:

assumes *finite A*

and $\text{card } (f \text{ ` } A) = \text{card } A$

shows *bij-betw f A (f ` A)*

$\langle \text{proof} \rangle$

Every bijective function between two finite subsets of a set S can be turned into a compatible renaming (with finite domain) on S .

lemma *bij-betw-extend*:

assumes *: *bij-betw f A B*

and $A \subseteq S$

and $B \subseteq S$

and *finite A*

shows $\exists g. \text{finite } \{x. g \ x \neq x\} \wedge$
 $(\forall x \in UNIV - (A \cup B). g \ x = x) \wedge$
 $(\forall x \in A. g \ x = f \ x) \wedge$
bij-betw g S S

$\langle \text{proof} \rangle$

lemma *concat-nth*:

assumes $m < \text{length } xs$ **and** $n < \text{length } (xs ! m)$

and $i = \text{sum-list } (\text{map length } (\text{take } m \text{ } xs)) + n$

shows $\text{concat } xs ! i = xs ! m ! n$

$\langle \text{proof} \rangle$

lemma *concat-nth-length*:

$i < \text{length } uss \implies j < \text{length } (uss ! i) \implies$

$\text{sum-list } (\text{map length } (\text{take } i \text{ } uss)) + j < \text{length } (\text{concat } uss)$

$\langle \text{proof} \rangle$

lemma *less-length-concat*:

assumes $i < \text{length } (\text{concat } xs)$

shows $\exists m \ n.$

$i = \text{sum-list } (\text{map length } (\text{take } m \text{ } xs)) + n \wedge$

$m < \text{length } xs \wedge n < \text{length } (xs ! m) \wedge \text{concat } xs ! i = xs ! m ! n$

$\langle \text{proof} \rangle$

lemma *concat-remove-nth*:

assumes $i < \text{length } sss$

and $j < \text{length } (sss ! i)$

defines $k \equiv \text{sum-list } (\text{map length } (\text{take } i \text{ } sss)) + j$

shows $\text{concat } (\text{take } i \text{ } sss @ \text{remove-nth } j \text{ } (sss ! i) \# \text{drop } (\text{Suc } i) \text{ } sss) = \text{remove-nth } k \text{ } (\text{concat } sss)$

$\langle \text{proof} \rangle$

lemma *nth-append-Cons*: $(xs @ y \# zs) ! i =$

$(\text{if } i < \text{length } xs \text{ then } xs ! i \text{ else if } i = \text{length } xs \text{ then } y \text{ else } zs ! (i - \text{Suc } (\text{length } xs)))$

$\langle \text{proof} \rangle$

lemma *finite-imp-eq* [simp]:

$\text{finite } \{x. P \ x \longrightarrow Q \ x\} \longleftrightarrow \text{finite } \{x. \neg P \ x\} \wedge \text{finite } \{x. Q \ x\}$

$\langle \text{proof} \rangle$

lemma *sum-list-take-eq*:

fixes $xs :: \text{nat list}$

shows $k < i \implies i < \text{length } xs \implies \text{sum-list } (\text{take } i \text{ } xs) =$

$\text{sum-list } (\text{take } k \text{ } xs) + xs ! k + \text{sum-list } (\text{take } (i - \text{Suc } k) \text{ } (\text{drop } (\text{Suc } k) \text{ } xs))$

$\langle \text{proof} \rangle$

lemma *nth-equalityE*:

$xs = ys \implies (\text{length } xs = \text{length } ys \implies (\bigwedge i. i < \text{length } xs \implies xs ! i = ys ! i) \implies P) \implies P$

$\langle \text{proof} \rangle$

fun *fold-map* :: $('a \Rightarrow 'b \Rightarrow 'c \times 'b) \Rightarrow 'a \text{ list} \Rightarrow 'b \Rightarrow 'c \text{ list} \times 'b$ **where**

$\text{fold-map } f \ [] \ y = ([], y)$

$| \text{fold-map } f \ (x \# xs) \ y = (\text{case } f \ x \ y \text{ of}$

$(x', y') \Rightarrow \text{case fold-map } f \text{ } xs \text{ } y' \text{ of}$
 $(xs', y'') \Rightarrow (x' \# xs', y'')$

lemma *fold-map-cong* [*fundef-cong*]:
assumes $a = b$ **and** $xs = ys$
and $\bigwedge x. x \in \text{set } xs \implies f x = g x$
shows $\text{fold-map } f \text{ } xs \text{ } a = \text{fold-map } g \text{ } ys \text{ } b$
 $\langle \text{proof} \rangle$

lemma *fold-map-map-conv*:
assumes $\bigwedge x \text{ } ys. x \in \text{set } xs \implies f (g x) (g' x @ ys) = (h x, ys)$
shows $\text{fold-map } f \text{ } (\text{map } g \text{ } xs) (\text{concat } (\text{map } g' \text{ } xs) @ ys) = (\text{map } h \text{ } xs, ys)$
 $\langle \text{proof} \rangle$

lemma *map-fst-fold-map*:
 $\text{map } f \text{ } (\text{fst } (\text{fold-map } g \text{ } xs \text{ } y)) = \text{fst } (\text{fold-map } (\lambda a \text{ } b. \text{apfst } f (g a b)) \text{ } xs \text{ } y)$
 $\langle \text{proof} \rangle$

1.1 Combinators

definition *const* :: $'a \Rightarrow 'b \Rightarrow 'a$ **where**
 $\text{const} \equiv (\lambda x \text{ } y. x)$

definition *flip* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c)$ **where**
 $\text{flip } f \equiv (\lambda x \text{ } y. f y x)$
declare *flip-def*[*simp*]

lemma *const-apply*[*simp*]: $\text{const } x \text{ } y = x$
 $\langle \text{proof} \rangle$

lemma *foldr-Cons-append-conv* [*simp*]:
 $\text{foldr } (\#) \text{ } xs \text{ } ys = xs @ ys$
 $\langle \text{proof} \rangle$

lemma *foldl-flip-Cons*[*simp*]:
 $\text{foldl } (\text{flip } (\#)) \text{ } xs \text{ } ys = \text{rev } ys @ xs$
 $\langle \text{proof} \rangle$

already present as *foldr-conv-foldl*, but direction seems odd

lemma *foldr-flip-rev*[*simp*]:
 $\text{foldr } (\text{flip } f) \text{ } (\text{rev } xs) \text{ } a = \text{foldl } f \text{ } a \text{ } xs$
 $\langle \text{proof} \rangle$

already present as *foldl-conv-foldr*, but direction seems odd

lemma *foldl-flip-rev*[*simp*]:
 $\text{foldl } (\text{flip } f) \text{ } a \text{ } (\text{rev } xs) = \text{foldr } f \text{ } xs \text{ } a$
 $\langle \text{proof} \rangle$

```
fun debug :: (String.literal  $\Rightarrow$  String.literal)  $\Rightarrow$  String.literal  $\Rightarrow$  'a  $\Rightarrow$  'a where
  debug i t x = x
```

1.2 Distinct Lists and Partitions

This theory provides some auxiliary lemmas related to lists with distinct elements and partitions. This is mainly used for dealing with *linear* terms.

lemma *distinct-alt*:

```
assumes  $\forall x. \text{length } (\text{filter } ((=) x) xs) \leq 1$ 
shows distinct xs
 $\langle \text{proof} \rangle$ 
```

lemma *distinct-filter2*:

```
assumes  $\forall i < \text{size } xs. \forall j < \text{size } xs. i \neq j \wedge f (xs!i) \wedge f (xs!j) \longrightarrow xs!i \neq xs!j$ 
shows distinct (filter f xs)
 $\langle \text{proof} \rangle$ 
```

lemma *distinct-is-partition*:

```
assumes distinct xs
shows is-partition (map ( $\lambda x. \{x\}$ ) xs)
 $\langle \text{proof} \rangle$ 
```

lemma *is-partition-append*:

```
assumes is-partition xs and is-partition zs
and  $\forall i < \text{length } xs. xs!i \cap \bigcup (\text{set } zs) = \{\}$ 
shows is-partition (xs@zs)
 $\langle \text{proof} \rangle$ 
```

lemma *distinct-is-partition-sets*:

```
assumes distinct xs
and  $xs = \text{concat } ys$ 
shows is-partition (map set ys)
 $\langle \text{proof} \rangle$ 
```

end

theory *Abstract-Rewriting-Impl*

imports

Abstract-Rewriting.Abstract-Rewriting

begin

partial-function (*option*) *compute-NF* :: ('a \Rightarrow 'a *option*) \Rightarrow 'a \Rightarrow 'a *option*

```
where [simp,code]: compute-NF f a = (case f a of None  $\Rightarrow$  Some a | Some b  $\Rightarrow$ 
  compute-NF f b)
```

lemma *compute-NF-sound*: **assumes** *res*: *compute-NF* f a = Some b

and *f-sound*: $\bigwedge a b. f a = \text{Some } b \implies (a,b) \in r$

shows $(a,b) \in r^*$

$\langle \text{proof} \rangle$

lemma *compute-NF-complete*: **assumes** *res*: *compute-NF f a = Some b*
and *f-complete*: $\bigwedge a. f\ a = \text{None} \implies a \in \text{NF } r$
shows $b \in \text{NF } r$
 $\langle \text{proof} \rangle$

lemma *compute-NF-SN*: **assumes** *SN*: *SN r*
and *f-sound*: $\bigwedge a\ b. f\ a = \text{Some } b \implies (a,b) \in r$
shows $\exists b. \text{compute-NF } f\ a = \text{Some } b$ (**is** *?P a*)
 $\langle \text{proof} \rangle$

definition *compute-trancl* $A\ R = R^{\wedge+}$ “ *A*
lemma *compute-trancl-rtrancl*[*code-unfold*]: $\{b. (a,b) \in R^{\wedge*}\} = \text{insert } a\ (\text{compute-trancl } \{a\}\ R)$
 $\langle \text{proof} \rangle$

lemma [*code*]: *compute-trancl A R = (let B = R “ A in*
if B \subseteq {} then {} else B \cup compute-trancl B { ab \in R . fst ab \notin A \wedge snd ab \notin
B})
 $\langle \text{proof} \rangle$

lemma [*code-unfold*]: $R^{\wedge+}$ “ *A = compute-trancl A R* $\langle \text{proof} \rangle$
lemma *compute-rtrancl*[*code-unfold*]: $R^{\wedge*}$ “ *A = A \cup compute-trancl A R*
 $\langle \text{proof} \rangle$
lemma [*code-unfold*]: $(a,b) \in R^{\wedge+} \longleftrightarrow b \in \text{compute-trancl } \{a\}\ R$ $\langle \text{proof} \rangle$
lemma [*code-unfold*]: $(a,b) \in R^{\wedge*} \longleftrightarrow b = a \vee b \in \text{compute-trancl } \{a\}\ R$
 $\langle \text{proof} \rangle$

end

2 Closure-Operations on Relations

theory *Relation-Closure*
imports *Abstract-Rewriting.Relative-Rewriting*
begin

locale *rel-closure* =
fixes *cop* :: $'b \Rightarrow 'a \Rightarrow 'a$ — closure operator
and *nil* :: $'b$
and *add* :: $'b \Rightarrow 'b \Rightarrow 'b$
assumes *cop-nil*: *cop nil x = x*
assumes *cop-add*: *cop (add a b) x = cop a (cop b x)*
begin

inductive-set *closure* **for** *r* :: $'a\ \text{rel}$
where
 $\langle \text{intro} \rangle$: $(x, y) \in r \implies (\text{cop } a\ x, \text{cop } a\ y) \in \text{closure } r$

lemma *closureI2*: $(x, y) \in r \implies u = \text{cop } a\ x \implies v = \text{cop } a\ y \implies (u, v) \in \text{closure}$

$r \langle \text{proof} \rangle$

lemma *closure-mono*: $r \subseteq s \implies \text{closure } r \subseteq \text{closure } s \langle \text{proof} \rangle$

lemma *subset-closure*: $r \subseteq \text{closure } r$
 $\langle \text{proof} \rangle$

definition *closed* $r \longleftrightarrow \text{closure } r \subseteq r$

lemma *closure-subset*: $\text{closed } r \implies \text{closure } r \subseteq r$
 $\langle \text{proof} \rangle$

lemma *closedI* [*Pure.intro*, *intro*]: $(\bigwedge x y a. (x, y) \in r \implies (\text{cop } a \ x, \text{cop } a \ y) \in r) \implies \text{closed } r$
 $\langle \text{proof} \rangle$

lemma *closedD* [*dest*]: $\text{closed } r \implies (x, y) \in r \implies (\text{cop } a \ x, \text{cop } a \ y) \in r$
 $\langle \text{proof} \rangle$

lemma *closed-closure* [*intro*]: $\text{closed } (\text{closure } r)$
 $\langle \text{proof} \rangle$

lemma *subset-closure-Un*:
 $\text{closure } r \subseteq \text{closure } (r \cup s)$
 $\text{closure } s \subseteq \text{closure } (r \cup s)$
 $\langle \text{proof} \rangle$

lemma *closure-Un*: $\text{closure } (r \cup s) = \text{closure } r \cup \text{closure } s$
 $\langle \text{proof} \rangle$

lemma *closure-id* [*simp*]: $\text{closed } r \implies \text{closure } r = r$
 $\langle \text{proof} \rangle$

lemma *closed-Un* [*intro*]: $\text{closed } r \implies \text{closed } s \implies \text{closed } (r \cup s) \langle \text{proof} \rangle$

lemma *closed-Inr* [*intro*]: $\text{closed } r \implies \text{closed } s \implies \text{closed } (r \cap s) \langle \text{proof} \rangle$

lemma *closed-rtrancI* [*intro*]: $\text{closed } r \implies \text{closed } (r^*)$
 $\langle \text{proof} \rangle$

lemma *closed-trancI* [*intro*]: $\text{closed } r \implies \text{closed } (r^+)$
 $\langle \text{proof} \rangle$

lemma *closed-converse* [*intro*]: $\text{closed } r \implies \text{closed } (r^{-1}) \langle \text{proof} \rangle$

lemma *closed-comp* [*intro*]: $\text{closed } r \implies \text{closed } s \implies \text{closed } (r \circ s) \langle \text{proof} \rangle$

lemma *closed-relpow* [*intro*]: $\text{closed } r \implies \text{closed } (r \smallfrown n)$
 $\langle \text{proof} \rangle$

```

lemma closed-conversion [intro]: closed  $r \implies \text{closed } (r^{\leftrightarrow*})$ 
   $\langle \text{proof} \rangle$ 

lemma closed-relto [intro]: closed  $r \implies \text{closed } s \implies \text{closed } (\text{relto } r \ s)$   $\langle \text{proof} \rangle$ 

lemma closure-diff-subset: closure  $r - \text{closure } s \subseteq \text{closure } (r - s)$   $\langle \text{proof} \rangle$ 

end

end

```

3 Term Rewriting

```

theory Term-Rewriting
  imports
    First-Order-Terms.Subterm-and-Context
    Relation-Closure
  begin

```

A rewrite rule is a pair of terms. A term rewrite system (TRS) is a set of rewrite rules.

```

type-synonym ( $'f$ ,  $'v$ ) rule = ( $'f$ ,  $'v$ ) term  $\times$  ( $'f$ ,  $'v$ ) term
type-synonym ( $'f$ ,  $'v$ ) trs = ( $'f$ ,  $'v$ ) rule set

```

```

inductive-set rstep ::  $- \Rightarrow ('f, 'v) \text{ term rel}$  for  $R :: ('f, 'v) \text{ trs}$ 
  where
    rstep:  $\bigwedge C \ \sigma \ l \ r. (l, r) \in R \implies s = C\langle l \cdot \sigma \rangle \implies t = C\langle r \cdot \sigma \rangle \implies (s, t) \in \text{rstep } R$ 

```

```

lemma rstep-induct-rule [case-names IH, induct set: rstep]:
  assumes  $(s, t) \in \text{rstep } R$ 
  and  $\bigwedge C \ \sigma \ l \ r. (l, r) \in R \implies P \ (C\langle l \cdot \sigma \rangle) \ (C\langle r \cdot \sigma \rangle)$ 
  shows  $P \ s \ t$ 
   $\langle \text{proof} \rangle$ 

```

An alternative induction scheme that treats the rule-case, the substitution-case, and the context-case separately.

```

lemma rstep-induct [consumes 1, case-names rule subst ctxt]:
  assumes  $(s, t) \in \text{rstep } R$ 
  and rule:  $\bigwedge l \ r. (l, r) \in R \implies P \ l \ r$ 
  and subst:  $\bigwedge s \ t \ \sigma. P \ s \ t \implies P \ (s \cdot \sigma) \ (t \cdot \sigma)$ 
  and ctxt:  $\bigwedge s \ t \ C. P \ s \ t \implies P \ (C\langle s \rangle) \ (C\langle t \rangle)$ 
  shows  $P \ s \ t$ 
   $\langle \text{proof} \rangle$ 

```

```

lemmas rstepI = rstep.intros [intro]

```


lemmas $rstepE = rstep.cases$ [elim]

lemma $rstep-ctxt$ [intro]: $(s, t) \in rstep\ R \implies (C\langle s \rangle, C\langle t \rangle) \in rstep\ R$
 ⟨proof⟩

lemma $rstep-rule$ [intro]: $(l, r) \in R \implies (l, r) \in rstep\ R$
 ⟨proof⟩

lemma $rstep-subst$ [intro]: $(s, t) \in rstep\ R \implies (s \cdot \sigma, t \cdot \sigma) \in rstep\ R$
 ⟨proof⟩

lemma $rstep-empty$ [simp]: $rstep\ \{\} = \{\}$
 ⟨proof⟩

lemma $rstep-mono$: $R \subseteq S \implies rstep\ R \subseteq rstep\ S$
 ⟨proof⟩

lemma $rstep-union$: $rstep\ (R \cup S) = rstep\ R \cup rstep\ S$
 ⟨proof⟩

lemma $rstep-converse$ [simp]: $rstep\ (R^{-1}) = (rstep\ R)^{-1}$
 ⟨proof⟩

interpretation $subst$: $rel-closure\ \lambda\sigma\ t.\ t \cdot \sigma\ Var\ \lambda x\ y.\ y \circ_s x$ ⟨proof⟩
declare $subst.closure.induct$ [consumes 1, case-names $subst$, induct pred: $subst.closure$]
declare $subst.closure.cases$ [consumes 1, case-names $subst$, cases pred: $subst.closure$]

interpretation $ctxt$: $rel-closure\ ctxt-apply-term\ \Box\ (\circ_c)$ ⟨proof⟩
declare $ctxt.closure.induct$ [consumes 1, case-names $ctxt$, induct pred: $ctxt.closure$]
declare $ctxt.closure.cases$ [consumes 1, case-names $ctxt$, cases pred: $ctxt.closure$]

lemma $rstep-eq-closure$: $rstep\ R = ctxt.closure\ (subst.closure\ R)$
 ⟨proof⟩

lemma $ctxt-closed-rstep$ [intro]: $ctxt.closed\ (rstep\ R)$
 ⟨proof⟩

lemma $ctxt-closed-one$:
 $ctxt.closed\ r \implies (s, t) \in r \implies (Fun\ f\ (ss\ @\ s\ \# \ ts), Fun\ f\ (ss\ @\ t\ \# \ ts)) \in r$
 ⟨proof⟩

lemma $args-steps-imp-steps$:
assumes $ctxt.closed\ r$
and len : $length\ ts = length\ us$
and $\forall i < length\ us.\ (ts\ !\ i, us\ !\ i) \in r^*$
shows $(Fun\ f\ ts, Fun\ f\ us) \in r^*$
 ⟨proof⟩

end

4 Term Rewrite Systems

```

theory Trs
  imports
    First-Order-Terms.Term-More
    Abstract-Rewriting.Relative-Rewriting
    Term-Rewriting
begin

```

4.1 Variables of Rules

```

definition
  vars-rule :: ('f, 'v) rule  $\Rightarrow$  'v set
where
  vars-rule r = vars-term (fst r)  $\cup$  vars-term (snd r)

```

```

lemma finite-vars-rule:
  finite (vars-rule r)
  <proof>

```

4.2 Variables of TRSs

```

definition vars-trs :: ('f, 'v) trs  $\Rightarrow$  'v set where
  vars-trs R = ( $\bigcup$  r $\in$ R. vars-rule r)

```

```

lemma vars-trs-union: vars-trs (R  $\cup$  S) = vars-trs R  $\cup$  vars-trs S
  <proof>

```

```

lemma finite-trs-has-finite-vars:
  assumes finite R shows finite (vars-trs R)
  <proof>

```

```

lemmas vars-defs = vars-trs-def vars-rule-def

```

4.3 Function Symbols of Rules

```

definition funs-rule :: ('f, 'v) rule  $\Rightarrow$  'f set where
  funs-rule r = funs-term (fst r)  $\cup$  funs-term (snd r)

```

The same including arities.

```

definition funas-rule :: ('f, 'v) rule  $\Rightarrow$  'f sig where
  funas-rule r = funas-term (fst r)  $\cup$  funas-term (snd r)

```

4.4 Function Symbols of TRSs

```

definition funs-trs :: ('f, 'v) trs  $\Rightarrow$  'f set where
  funs-trs R = ( $\bigcup$  r $\in$ R. funs-rule r)

```

definition *funas-trs* :: (*f*, *v*) *trs* \Rightarrow *f sig* **where**

funas-trs *R* = ($\bigcup_{r \in R. \text{funas-rule } r}$)

lemma *funas-rule-funas-rule*: *funas-rule* *rl* = *fst* ‘ *funas-rule* *rl*
 <proof>

lemma *funas-trs-funas-trs*: *funas-trs* *R* = *fst* ‘ *funas-trs* *R*
 <proof>

lemma *finite-funas-rule*: *finite* (*funas-rule* *lr*)
 <proof>

lemma *finite-funas-trs*:
 assumes *finite* *R*
 shows *finite* (*funas-trs* *R*)
 <proof>

lemma *funas-empty[simp]*: *funas-trs* {} = {} <proof>

lemma *funas-trs-union[simp]*: *funas-trs* (*R* \cup *S*) = *funas-trs* *R* \cup *funas-trs* *S*
 <proof>

definition *funas-args-rule* :: (*f*, *v*) *rule* \Rightarrow *f sig* **where**
funas-args-rule *r* = *funas-args-term* (*fst* *r*) \cup *funas-args-term* (*snd* *r*)

definition *funas-args-trs* :: (*f*, *v*) *trs* \Rightarrow *f sig* **where**
funas-args-trs *R* = ($\bigcup_{r \in R. \text{funas-args-rule } r}$)

lemmas *funas-args-defs* =
funas-args-trs-def *funas-args-rule-def* *funas-args-term-def*

definition *roots-rule* :: (*f*, *v*) *rule* \Rightarrow *f sig*
where
roots-rule *r* = *set-option* (*root* (*fst* *r*)) \cup *set-option* (*root* (*snd* *r*))

definition *roots-trs* :: (*f*, *v*) *trs* \Rightarrow *f sig* **where**
roots-trs *R* = ($\bigcup_{r \in R. \text{roots-rule } r}$)

lemmas *roots-defs* =
roots-trs-def *roots-rule-def*

definition *funas-head* :: (*f*, *v*) *trs* \Rightarrow (*f*, *v*) *trs* \Rightarrow *f sig* **where**
funas-head *P* *R* = *funas-trs* *P* - (*funas-trs* *R* \cup *funas-args-trs* *P*)

lemmas *funas-defs* = *funas-trs-def* *funas-rule-def*

lemmas *funas-defs* =
funas-trs-def *funas-rule-def*
funas-args-defs

funas-head-def
roots-defs

A function symbol is said to be *defined* (w.r.t. to a given TRS) if it occurs as root of some left-hand side.

definition

$defined :: ('f, 'v) trs \Rightarrow ('f \times nat) \Rightarrow bool$

where

$defined\ R\ fn \longleftrightarrow (\exists l\ r. (l, r) \in R \wedge root\ l = Some\ fn)$

lemma *defined-funas-trs*: **assumes** *d*: *defined* *R* *fn* **shows** *fn* $\in funas-trs\ R$
 $\langle proof \rangle$

lemma *ctxt-closed-R-imp-supt-R-distr*:

assumes *ctxt.closed* *R* **and** $s \triangleright t$ **and** $(t, u) \in R$ **shows** $\exists t. (s, t) \in R \wedge t \triangleright u$
 $\langle proof \rangle$

lemma *ctxt-closed-imp-qc-supt*: *ctxt.closed* *R* $\implies \{\triangleright\} \circ R \subseteq R \circ (R \cup \{\triangleright\})^*$
 $\langle proof \rangle$

Let *R* be a relation on terms that is closed under contexts. If *R* is well-founded then $R \cup \triangleright$ is well-founded.

lemma *SN-imp-SN-union-supt*:

assumes *SN* *R* **and** *ctxt.closed* *R*
shows *SN* $(R \cup \{\triangleright\})$
 $\langle proof \rangle$

lemma *stable-loop-imp-not-SN*:

assumes *stable*: *subst.closed* *r* **and** *steps*: $(s, s \cdot \sigma) \in r^+$
shows $\neg SN-on\ r\ \{s\}$
 $\langle proof \rangle$

lemma *subst-closed-supteq*: *subst.closed* $\{\triangleright\}$ $\langle proof \rangle$

lemma *subst-closed-supt*: *subst.closed* $\{\triangleright\}$ $\langle proof \rangle$

lemma *rstep-relcomp-idemp1* [*simp*]:

$rstep\ (rstep\ R\ O\ rstep\ S) = rstep\ R\ O\ rstep\ S$
 $\langle proof \rangle$

lemma *rstep-relcomp-idemp2* [*simp*]:

$rstep\ (rstep\ R\ O\ rstep\ S\ O\ rstep\ T) = rstep\ R\ O\ rstep\ S\ O\ rstep\ T$
 $\langle proof \rangle$

lemma *ctxt-closed-rsteps* [*intro*]: *ctxt.closed* $((rstep\ R)^*)$ $\langle proof \rangle$

lemma *subset-rstep*: $R \subseteq rstep\ R$ $\langle proof \rangle$

lemma *subst-closure-rstep-subset*: *subst.closure* $(rstep\ R) \subseteq rstep\ R$

$\langle \text{proof} \rangle$

lemma *subst-closed-rstep* [intro]: *subst.closed* (*rstep* *R*) $\langle \text{proof} \rangle$

lemma *subst-closed-rsteps*: *subst.closed* ((*rstep* *R*)^{*}) $\langle \text{proof} \rangle$

lemma *ctxt-closed-supt-subset*: *ctxt.closed* *R* $\implies \{\triangleright\} \ O \ R \subseteq R \ O \ \{\triangleright\}$ $\langle \text{proof} \rangle$

lemmas *supt-rsteps-subset* = *ctxt-closed-supt-subset* [OF *ctxt-closed-rsteps*]

lemma *supteq-rsteps-subset*:

$\{\triangleright\} \ O \ (\text{rstep } R)^* \subseteq (\text{rstep } R)^* \ O \ \{\triangleright\}$ (is ?*S* \subseteq ?*T*)
 $\langle \text{proof} \rangle$

lemma *quasi-commute-rsteps-supt*:

quasi-commute ((*rstep* *R*)^{*}) $\{\triangleright\}$
 $\langle \text{proof} \rangle$

lemma *rstep-UN*:

rstep ($\bigcup_{i \in A} R \ i$) = ($\bigcup_{i \in A} \text{rstep } (R \ i)$)
 $\langle \text{proof} \rangle$

definition

rstep-r-p-s :: (*'f*, *'v*) *trs* \Rightarrow (*'f*, *'v*) *rule* \Rightarrow *pos* \Rightarrow (*'f*, *'v*) *subst* \Rightarrow (*'f*, *'v*) *trs*

where

rstep-r-p-s *R* *r* *p* σ = {(*s*, *t*).

let *C* = *ctxt-of-pos-term* *p* *s* in *p* \in *poss* *s* \wedge *r* \in *R* \wedge (*C* $\langle \text{fst } r \cdot \sigma \rangle$ = *s*) \wedge
(*C* $\langle \text{snd } r \cdot \sigma \rangle$ = *t*)}

lemma *rstep-r-p-s-def'*:

rstep-r-p-s *R* *r* *p* σ = {(*s*, *t*).
p \in *poss* *s* \wedge *r* \in *R* \wedge *s* | - *p* = *fst* *r* \cdot σ \wedge *t* = *replace-at* *s* *p* (*snd* *r* \cdot σ)} (is ?*l*
= ?*r*)
 $\langle \text{proof} \rangle$

lemma *parallel-steps*:

fixes *p*₁ :: *pos*

assumes (*s*, *t*) \in *rstep-r-p-s* *R*₁ (*l*₁, *r*₁) *p*₁ $\sigma₁$

and (*s*, *u*) \in *rstep-r-p-s* *R*₂ (*l*₂, *r*₂) *p*₂ σ ₂

and *par*: *p*₁ \perp *p*₂

shows (*t*, (*ctxt-of-pos-term* *p*₁ *u*) $\langle r_1 \cdot \sigma_1 \rangle$) \in *rstep-r-p-s* *R*₂ (*l*₂, *r*₂) *p*₂ σ ₂ \wedge

(*u*, (*ctxt-of-pos-term* *p*₁ *u*) $\langle r_1 \cdot \sigma_1 \rangle$) \in *rstep-r-p-s* *R*₁ (*l*₁, *r*₁) *p*₁ σ ₁

$\langle \text{proof} \rangle$

lemma *rstep-iff-rstep-r-p-s*:

(*s*, *t*) \in *rstep* *R* $\iff (\exists l \ r \ p \ \sigma. (s, t) \in \text{rstep-r-p-s } R \ (l, r) \ p \ \sigma)$ (is ?*lhs* = ?*rhs*)
 $\langle \text{proof} \rangle$

lemma *rstep-r-p-s-imp-rstep*:

assumes $(s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ p\ \sigma$
shows $(s, t) \in rstep\ R$
 $\langle proof \rangle$

Rewriting steps below the root position.

definition

$nrrstep :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs$

where

$nrrstep\ R = \{(s, t). \exists r\ i\ ps\ \sigma. (s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ (i\#ps)\ \sigma\}$

An alternative characterisation of non-root rewrite steps.

lemma *nrrstep-def'*:

$nrrstep\ R = \{(s, t). \exists l\ r\ C\ \sigma. (l, r) \in R \wedge C \neq \square \wedge s = C\langle l \cdot \sigma \rangle \wedge t = C\langle r \cdot \sigma \rangle\}$
(is ?lhs = ?rhs)

$\langle proof \rangle$

lemma *nrrstepI*: $(l, r) \in R \Longrightarrow s = C\langle l \cdot \sigma \rangle \Longrightarrow t = C\langle r \cdot \sigma \rangle \Longrightarrow C \neq \square \Longrightarrow (s, t) \in nrrstep\ R$ $\langle proof \rangle$

lemma *nrrstep-union*: $nrrstep\ (R \cup S) = nrrstep\ R \cup nrrstep\ S$
 $\langle proof \rangle$

lemma *nrrstep-empty[simp]*: $nrrstep\ \{\} = \{\}$ $\langle proof \rangle$

Rewriting step at the root position.

definition

$rrstep :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs$

where

$rrstep\ R = \{(s, t). \exists r\ \sigma. (s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ []\ \sigma\}$

An alternative characterisation of root rewrite steps.

lemma *rrstep-def'*: $rrstep\ R = \{(s, t). \exists l\ r\ \sigma. (l, r) \in R \wedge s = l \cdot \sigma \wedge t = r \cdot \sigma\}$
(is - = ?rhs)

$\langle proof \rangle$

lemma *rules-subset-rrstep [simp]*: $R \subseteq rrstep\ R$
 $\langle proof \rangle$

lemma *rrstep-union*: $rrstep\ (R \cup S) = rrstep\ R \cup rrstep\ S$ $\langle proof \rangle$

lemma *rrstep-empty[simp]*: $rrstep\ \{\} = \{\}$
 $\langle proof \rangle$

lemma *subst-closed-rrstep*: $subst.closed\ (rrstep\ R)$
 $\langle proof \rangle$

lemma *rstep-iff-rrstep-or-nrrstep*: $rstep\ R = (rrstep\ R \cup nrrstep\ R)$

$\langle \text{proof} \rangle$

lemma *rstep-i-pos-imp-rstep-arg-i-pos*:

assumes *nrrstep*: $(\text{Fun } f \text{ } ss, t) \in \text{rstep-r-p-s } R \text{ } (l, r) \text{ } (i \# ps) \text{ } \sigma$

shows $(ss!i, t[-i]) \in \text{rstep-r-p-s } R \text{ } (l, r) \text{ } ps \text{ } \sigma$

$\langle \text{proof} \rangle$

lemma *ctxt-closure-rstep-eq [simp]*: $\text{ctxt.closure } (\text{rstep } R) = \text{rstep } R$

$\langle \text{proof} \rangle$

lemma *subst-closure-rstep-eq [simp]*: $\text{subst.closure } (\text{rstep } R) = \text{rstep } R$

$\langle \text{proof} \rangle$

lemma *supt-rstep-subset*:

$\{\triangleright\} \text{ } O \text{ } \text{rstep } R \subseteq \text{rstep } R \text{ } O \text{ } \{\triangleright\}$

$\langle \text{proof} \rangle$

lemma *ne-rstep-seq-imp-list-of-terms*:

assumes $(s, t) \in (\text{rstep } R)^+$

shows $\exists ts. \text{length } ts > 1 \wedge ts!0 = s \wedge ts!(\text{length } ts - 1) = t \wedge$

$(\forall i < \text{length } ts - 1. (ts!i, ts!(\text{Suc } i)) \in (\text{rstep } R)) \text{ (is } \exists ts. - \wedge - \wedge - \wedge ?P \text{ } ts)$

$\langle \text{proof} \rangle$

locale *E-compatible* =

fixes $R :: ('f, 'v) \text{trs}$ **and** $E :: ('f, 'v) \text{trs}$

assumes $E: E \text{ } O \text{ } R = R \text{ } Id \subseteq E$

begin

definition *restrict-SN-supt-E* :: $('f, 'v) \text{trs}$ **where**

$\text{restrict-SN-supt-E} = \text{restrict-SN } R \text{ } R \cup \text{restrict-SN } (E \text{ } O \text{ } \{\triangleright\} \text{ } O \text{ } E) \text{ } R$

lemma *ctxt-closed-R-imp-supt-restrict-SN-E-distr*:

assumes *ctxt.closed* R

and $(s, t) \in (\text{restrict-SN } (E \text{ } O \text{ } \{\triangleright\}) \text{ } R)$

and $(t, u) \in \text{restrict-SN } R \text{ } R$

shows $(\exists t. (s, t) \in \text{restrict-SN } R \text{ } R \wedge (t, u) \in \text{restrict-SN } (E \text{ } O \text{ } \{\triangleright\}) \text{ } R) \text{ (is } \exists t. - \wedge (t, u) \in ?\text{snSub})$

$\langle \text{proof} \rangle$

lemma *ctxt-closed-R-imp-restrict-SN-qc-E-supt*:

assumes *ctxt.closed* R

shows $\text{quasi-commute } (\text{restrict-SN } R \text{ } R) \text{ } (\text{restrict-SN } (E \text{ } O \text{ } \{\triangleright\} \text{ } O \text{ } E) \text{ } R) \text{ (is quasi-commute } ?r \text{ } ?s)$

$\langle \text{proof} \rangle$

lemma *ctxt-closed-imp-SN-restrict-SN-E-supt*:

assumes *ctxt.closed* R

and $SN: SN \text{ } (E \text{ } O \text{ } \{\triangleright\} \text{ } O \text{ } E)$

shows $SN \text{ restrict-SN-supt-E}$
 $\langle proof \rangle$
end

lemma $E\text{-compatible-Id}$: $E\text{-compatible } R \text{ Id}$
 $\langle proof \rangle$

definition $restrict\text{-SN-supt} :: ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ trs}$ **where**
 $restrict\text{-SN-supt } R = restrict\text{-SN } R \ R \cup restrict\text{-SN } \{\triangleright\} \ R$

lemma $ctxt\text{-closed-SN-on-subt}$:
assumes $ctxt.\text{closed } R$ **and** $SN\text{-on } R \ \{s\}$ **and** $s \supseteq t$
shows $SN\text{-on } R \ \{t\}$
 $\langle proof \rangle$

lemma $ctxt\text{-closed-R-imp-supt-restrict-SN-distr}$:
assumes $R: ctxt.\text{closed } R$
and $st: (s,t) \in (restrict\text{-SN } \{\triangleright\} \ R)$
and $tu: (t,u) \in restrict\text{-SN } R \ R$
shows $(\exists t. (s,t) \in restrict\text{-SN } R \ R \wedge (t,u) \in restrict\text{-SN } \{\triangleright\} \ R) \text{ (is } \exists t. - \wedge (t,u) \in ?snSub)$
 $\langle proof \rangle$

lemma $ctxt\text{-closed-R-imp-restrict-SN-qc-supt}$:
assumes $ctxt.\text{closed } R$
shows $quasi\text{-commute } (restrict\text{-SN } R \ R) \ (restrict\text{-SN } supt \ R) \text{ (is } quasi\text{-commute } ?r \ ?s)$
 $\langle proof \rangle$

lemma $ctxt\text{-closed-imp-SN-restrict-SN-supt}$:
assumes $ctxt.\text{closed } R$
shows $SN \ (restrict\text{-SN-supt } R)$
 $\langle proof \rangle$

lemma $SN\text{-restrict-SN-supt-rstep}$:
shows $SN \ (restrict\text{-SN-supt } (rstep \ R))$
 $\langle proof \rangle$

lemma $nrrstep\text{-imp-pos-term}$:
 $(Fun \ f \ ss, t) \in nrrstep \ R \Longrightarrow$
 $\exists i \ s. \ t = Fun \ f \ (ss[i:=s]) \wedge (ss!i, s) \in rstep \ R \wedge i < length \ ss$
 $\langle proof \rangle$

lemma $rstep\text{-cases}$ [$consumes \ 1, case\text{-names } root \ nonroot$]:
 $\llbracket (s,t) \in rstep \ R; (s,t) \in nrrstep \ R \Longrightarrow P; (s,t) \in nrrstep \ R \Longrightarrow P \rrbracket \Longrightarrow P$
 $\langle proof \rangle$

lemma $nrrstep\text{-imp-rstep}$: $(s,t) \in nrrstep \ R \Longrightarrow (s,t) \in rstep \ R$

$\langle \text{proof} \rangle$

lemma *nrrstep-imp-Fun*: $(s, t) \in \text{nrrstep } R \implies \exists f \text{ ss. } s = \text{Fun } f \text{ ss}$
 $\langle \text{proof} \rangle$

lemma *nrrstep-imp-subt-rstep*:
assumes $(s, t) \in \text{nrrstep } R$
shows $\exists i. i < \text{num-args } s \wedge \text{num-args } s = \text{num-args } t \wedge (s|-[i], t|-[i]) \in \text{rstep } R$
 $\wedge (\forall j. i \neq j \longrightarrow s|-[j] = t|-[j])$
 $\langle \text{proof} \rangle$

lemma *finite-range*:
fixes $f :: \text{nat} \Rightarrow 'a$
assumes *finite* (*range* f)
shows $\exists x. \exists_{\infty} i. f \ i = x$
 $\langle \text{proof} \rangle$

lemma *nrrstep-subt*: **assumes** $(s, t) \in \text{nrrstep } R$ **shows** $\exists u \triangleleft s. \exists v \triangleleft t. (u, v) \in \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma *nrrstep-args*:
assumes $(s, t) \in \text{nrrstep } R$
shows $\exists f \text{ ss } ts. s = \text{Fun } f \text{ ss} \wedge t = \text{Fun } f \text{ ts} \wedge \text{length } ss = \text{length } ts$
 $\wedge (\exists j < \text{length } ss. (ss!j, ts!j) \in \text{rstep } R \wedge (\forall i < \text{length } ss. i \neq j \longrightarrow ss!i = ts!i))$
 $\langle \text{proof} \rangle$

lemma *nrrstep-iff-arg-rstep*:
 $(s, t) \in \text{nrrstep } R \longleftrightarrow$
 $(\exists f \text{ ss } i \ t'. s = \text{Fun } f \text{ ss} \wedge i < \text{length } ss \wedge t = \text{Fun } f \ (ss[i:=t']) \wedge (ss!i, t') \in \text{rstep } R)$
(is ?L \longleftrightarrow ?R)
 $\langle \text{proof} \rangle$

lemma *subterms-NF-imp-SN-on-nrrstep*:
assumes $\forall s \triangleleft t. s \in \text{NF } (\text{rstep } R)$ **shows** *SN-on* (*nrrstep* R) $\{t\}$
 $\langle \text{proof} \rangle$

lemma *args-NF-imp-SN-on-nrrstep*:
assumes $\forall t \in \text{set } ts. t \in \text{NF } (\text{rstep } R)$ **shows** *SN-on* (*nrrstep* R) $\{\text{Fun } f \text{ ts}\}$
 $\langle \text{proof} \rangle$

lemma *rrstep-imp-rule-subst*:
assumes $(s, t) \in \text{rrstep } R$
shows $\exists l \ r \ \sigma. (l, r) \in R \wedge (l \cdot \sigma) = s \wedge (r \cdot \sigma) = t$
 $\langle \text{proof} \rangle$

lemma *nrrstep-preserves-root*:

assumes $(Fun\ f\ ss, t) \in nrrstep\ R$ **(is** $(?s, t) \in nrrstep\ R$) **shows** $\exists ts. t = (Fun\ f\ ts)$
 $\langle proof \rangle$

lemma *nrrstep-equiv-root*: **assumes** $(s, t) \in nrrstep\ R$ **shows** $\exists f\ ss\ ts. s = Fun\ f\ ss \wedge t = Fun\ f\ ts$
 $\langle proof \rangle$

lemma *nrrstep-reflects-root*:
assumes $(s, Fun\ g\ ts) \in nrrstep\ R$ **(is** $(s, ?t) \in nrrstep\ R$)
shows $\exists ss. s = (Fun\ g\ ss)$
 $\langle proof \rangle$

lemma *nrrsteps-preserve-root*:
assumes $(Fun\ f\ ss, t) \in (nrrstep\ R)^*$
shows $\exists ts. t = (Fun\ f\ ts)$
 $\langle proof \rangle$

lemma *nrrstep-Fun-imp-arg-rsteps*:
assumes $(Fun\ f\ ss, Fun\ f\ ts) \in nrrstep\ R$ **(is** $(?s, ?t) \in nrrstep\ R$) **and** $i < length\ ss$
shows $(ss!i, ts!i) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *nrrstep-imp-arg-rsteps*:
assumes $(s, t) \in nrrstep\ R$ **and** $i < num-args\ s$ **shows** $(args\ s!i, args\ t!i) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *nrrsteps-imp-rsteps*: $(s, t) \in (nrrstep\ R)^* \implies (s, t) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *nrrstep-Fun-preserves-num-args*:
assumes $(Fun\ f\ ss, Fun\ f\ ts) \in nrrstep\ R$ **(is** $(?s, ?t) \in nrrstep\ R$)
shows $length\ ss = length\ ts$
 $\langle proof \rangle$

lemma *nrrstep-equiv-num-args*:
assumes $(s, t) \in nrrstep\ R$ **shows** $num-args\ s = num-args\ t$
 $\langle proof \rangle$

lemma *nrrsteps-equiv-num-args*:
assumes $(s, t) \in (nrrstep\ R)^*$ **shows** $num-args\ s = num-args\ t$
 $\langle proof \rangle$

lemma *nrrstep-preserves-num-args*:
assumes $(s, t) \in nrrstep\ R$ **and** $i < num-args\ s$ **shows** $i < num-args\ t$
 $\langle proof \rangle$

lemma *nrrstep-reflects-num-args*:

assumes $(s,t) \in \text{nrrstep } R$ **and** $i < \text{num-args } t$ **shows** $i < \text{num-args } s$
 $\langle \text{proof} \rangle$

lemma *nrrsteps-imp-arg-rsteps*:

assumes $(s,t) \in (\text{nrrstep } R)^*$ **and** $i < \text{num-args } s$
shows $(\text{args } s!i, \text{args } t!i) \in (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *nrrsteps-imp-eq-root-arg-rsteps*:

assumes $\text{steps}: (s,t) \in (\text{nrrstep } R)^*$
shows $\text{root } s = \text{root } t \wedge (\forall i < \text{num-args } s. (s \mid - [i], t \mid - [i]) \in (\text{rstep } R)^*)$
 $\langle \text{proof} \rangle$

definition

$\text{wf-trs} :: ('f, 'v) \text{trs} \Rightarrow \text{bool}$

where

$\text{wf-trs } R = (\forall l \ r. (l,r) \in R \longrightarrow (\exists f \ ts. l = \text{Fun } f \ ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l)$

lemma *wf-trs-imp-lhs-Fun*:

$\text{wf-trs } R \Longrightarrow (l,r) \in R \Longrightarrow \exists f \ ts. l = \text{Fun } f \ ts$
 $\langle \text{proof} \rangle$

lemma *rstep-imp-Fun*:

assumes $\text{wf-trs } R$
shows $(s, t) \in \text{rstep } R \Longrightarrow \exists f \ ss. s = \text{Fun } f \ ss$
 $\langle \text{proof} \rangle$

lemma *SN-on-imp-SN-on-subt*:

assumes $\text{SN-on } (\text{rstep } R) \{t\}$ **shows** $\forall s \sqsubseteq t. \text{SN-on } (\text{rstep } R) \{s\}$
 $\langle \text{proof} \rangle$

lemma *not-SN-on-subt-imp-not-SN-on*:

assumes $\neg \text{SN-on } (\text{rstep } R) \{t\}$ **and** $s \sqsupseteq t$
shows $\neg \text{SN-on } (\text{rstep } R) \{s\}$
 $\langle \text{proof} \rangle$

lemma *SN-on-instance-imp-SN-on-var*:

assumes $\text{SN-on } (\text{rstep } R) \{t \cdot \sigma\}$ **and** $x \in \text{vars-term } t$
shows $\text{SN-on } (\text{rstep } R) \{\text{Var } x \cdot \sigma\}$
 $\langle \text{proof} \rangle$

lemma *var-imp-var-of-arg*:

assumes $x \in \text{vars-term } (\text{Fun } f \ ss)$ **(is** $x \in \text{vars-term } ?s)$
shows $\exists i < \text{num-args } (\text{Fun } f \ ss). x \in \text{vars-term } (ss!i)$
 $\langle \text{proof} \rangle$

lemma *subt-instance-and-not-subst-imp-subt*:

$s \cdot \sigma \succeq t \implies \forall x \in \text{vars-term } s. \neg((\text{Var } x) \cdot \sigma \succeq t) \implies \exists u. s \succeq u \wedge t = u \cdot \sigma$
 $\langle \text{proof} \rangle$

lemma *SN-imp-SN-subt*:

$\text{SN-on } (\text{rstep } R) \{s\} \implies s \succeq t \implies \text{SN-on } (\text{rstep } R) \{t\}$
 $\langle \text{proof} \rangle$

lemma *subterm-preserves-SN-gen*:

assumes *ctxt*: *ctxt.closed* *R*
and *SN*: *SN-on* *R* $\{t\}$ **and** *supt*: $t \triangleright s$
shows *SN-on* *R* $\{s\}$
 $\langle \text{proof} \rangle$

context *E-compatible*
begin

lemma *SN-on-step-E-imp-SN-on*: **assumes** *SN-on* *R* $\{s\}$
and $(s, t) \in E$
shows *SN-on* *R* $\{t\}$
 $\langle \text{proof} \rangle$

lemma *SN-on-step-REs-imp-SN-on*:
assumes *R*: *ctxt.closed* *R*
and *st*: $(s, t) \in (R \cup E \circ \{\triangleright\} \circ E)$
and *SN*: *SN-on* *R* $\{s\}$
shows *SN-on* *R* $\{t\}$
 $\langle \text{proof} \rangle$

lemma *restrict-SN-supt-E-I*:

$\text{ctxt.closed } R \implies \text{SN-on } R \{s\} \implies (s, t) \in R \cup E \circ \{\triangleright\} \circ E \implies (s, t) \in$
 $\text{restrict-SN-supt-E}$
 $\langle \text{proof} \rangle$

lemma *ctxt-closed-imp-SN-on-E-supt*:

assumes *R*: *ctxt.closed* *R*
and *SN*: *SN* $(E \circ \{\triangleright\} \circ E)$
shows *SN-on* $(R \cup E \circ \{\triangleright\} \circ E) \{t. \text{SN-on } R \{t\}\}$
 $\langle \text{proof} \rangle$
end

lemma *subterm-preserves-SN*:

$\text{SN-on } (\text{rstep } R) \{t\} \implies t \triangleright s \implies \text{SN-on } (\text{rstep } R) \{s\}$
 $\langle \text{proof} \rangle$

lemma *SN-on-r-imp-SN-on-supt-union-r*:

assumes *ctxt*: *ctxt.closed* *R*
and *SN-on* *R* *T*

shows $SN\text{-on } (supt \cup R) \ T \text{ (is } SN\text{-on } ?S \ T)$
 $\langle proof \rangle$

lemma $SN\text{-on-rstep-imp-}SN\text{-on-supt-union-rstep}$:
 $SN\text{-on } (rstep \ R) \ T \implies SN\text{-on } (supt \cup rstep \ R) \ T$
 $\langle proof \rangle$

lemma $SN\text{-supt-r-trancl}$:
assumes $ctxt: ctxt.closed \ R$
and $a: SN \ R$
shows $SN \ ((supt \cup R)^+)$
 $\langle proof \rangle$

lemma $SN\text{-supt-rstep-trancl}$:
 $SN \ (rstep \ R) \implies SN \ ((supt \cup rstep \ R)^+)$
 $\langle proof \rangle$

lemma $SN\text{-imp-}SN\text{-arg-gen}$:
assumes $ctxt: ctxt.closed \ R$
and $SN: SN\text{-on } R \ \{Fun \ f \ ts\}$ **and** $arg: t \in set \ ts$ **shows** $SN\text{-on } R \ \{t\}$
 $\langle proof \rangle$

lemma $SN\text{-imp-}SN\text{-arg}$:
 $SN\text{-on } (rstep \ R) \ \{Fun \ f \ ts\} \implies t \in set \ ts \implies SN\text{-on } (rstep \ R) \ \{t\}$
 $\langle proof \rangle$

lemma $SN\text{-instance-imp-}SN$:
assumes $SN\text{-on } (rstep \ R) \ \{t \cdot \sigma\}$
shows $SN\text{-on } (rstep \ R) \ \{t\}$
 $\langle proof \rangle$

lemma $rstep\text{-imp-}C\text{-s-r}$:
assumes $(s, t) \in rstep \ R$
shows $\exists C \ \sigma \ l \ r. (l, r) \in R \wedge s = C \langle l \cdot \sigma \rangle \wedge t = C \langle r \cdot \sigma \rangle$
 $\langle proof \rangle$

lemma $SN\text{-Var}$:
assumes $wf\text{-trs } R$ **shows** $SN\text{-on } (rstep \ R) \ \{Var \ x\}$
 $\langle proof \rangle$

fun $map\text{-funs-rule} :: ('f \Rightarrow 'g) \Rightarrow ('f, 'v) \text{ rule} \Rightarrow ('g, 'v) \text{ rule}$
where
 $map\text{-funs-rule} \ fg \ lr = (map\text{-funs-term} \ fg \ (fst \ lr), map\text{-funs-term} \ fg \ (snd \ lr))$

fun $map\text{-funs-trs} :: ('f \Rightarrow 'g) \Rightarrow ('f, 'v) \text{ trs} \Rightarrow ('g, 'v) \text{ trs}$
where
 $map\text{-funs-trs} \ fg \ R = map\text{-funs-rule} \ fg \ ` \ R$

lemma *map-funs-trs-union*: $\text{map-funs-trs } fg \ (R \cup S) = \text{map-funs-trs } fg \ R \cup \text{map-funs-trs } fg \ S$

<proof>

lemma *rstep-map-funs-term*: **assumes** $R: \bigwedge f. f \in \text{funs-trs } R \implies h \ f = f$ **and**

step: $(s, t) \in \text{rstep } R$

shows $(\text{map-funs-term } h \ s, \text{map-funs-term } h \ t) \in \text{rstep } R$

<proof>

lemma *wf-trs-map-funs-trs[simp]*: $\text{wf-trs } (\text{map-funs-trs } f \ R) = \text{wf-trs } R$

<proof>

lemma *map-funs-trs-comp*: $\text{map-funs-trs } fg \ (\text{map-funs-trs } gh \ R) = \text{map-funs-trs } (fg \ o \ gh) \ R$

<proof>

lemma *map-funs-trs-mono*: **assumes** $R \subseteq R'$ **shows** $\text{map-funs-trs } fg \ R \subseteq \text{map-funs-trs } fg \ R'$

<proof>

lemma *map-funs-trs-power-mono*:

fixes $R \ R' :: ('f, 'v)\text{trs}$ **and** $fg :: 'f \Rightarrow 'f$

assumes $R \subseteq R'$ **shows** $((\text{map-funs-trs } fg)^{\sim n}) \ R \subseteq ((\text{map-funs-trs } fg)^{\sim n}) \ R'$

<proof>

declare *map-funs-trs.simps[simp del]*

lemma *rstep-imp-map-rstep*:

assumes $(s, t) \in \text{rstep } R$

shows $(\text{map-funs-term } fg \ s, \text{map-funs-term } fg \ t) \in \text{rstep } (\text{map-funs-trs } fg \ R)$

<proof>

lemma *rsteps-imp-map-rsteps*: **assumes** $(s, t) \in (\text{rstep } R)^*$

shows $(\text{map-funs-term } fg \ s, \text{map-funs-term } fg \ t) \in (\text{rstep } (\text{map-funs-trs } fg \ R))^*$

<proof>

lemma *SN-map-imp-SN*:

assumes $SN: SN\text{-on } (\text{rstep } (\text{map-funs-trs } fg \ R)) \ \{\text{map-funs-term } fg \ t\}$

shows $SN\text{-on } (\text{rstep } R) \ \{t\}$

<proof>

lemma *rstep-iff-map-rstep*:

assumes $\text{inj } fg$

shows $(s, t) \in \text{rstep } R \longleftrightarrow (\text{map-funs-term } fg \ s, \text{map-funs-term } fg \ t) \in \text{rstep } (\text{map-funs-trs } fg \ R)$

<proof>

lemma *rstep-map-funs-trs-power-mono*:

fixes $R \ R' :: ('f, 'v)\text{trs}$ **and** $fg :: 'f \Rightarrow 'f$

assumes *subset*: $R \subseteq R'$ **shows** $rstep\ ((map-funs-trs\ fg) \sim_n R) \subseteq rstep\ (((map-funs-trs\ fg) \sim_n) R')$
 $\langle proof \rangle$

lemma *subsetI3*: $(\bigwedge x\ y\ z. (x, y, z) \in A \implies (x, y, z) \in B) \implies A \subseteq B$ $\langle proof \rangle$

lemma *aux*: $(\bigcup a \in P. \{(x, y, z). x = fst\ a \wedge y = snd\ a \wedge Q\ a\ z\}) = \{(x, y, z). (x, y) \in P \wedge Q\ (x, y)\ z\}$ (**is** $?P = ?Q$)
 $\langle proof \rangle$

lemma *finite-imp-finite-DP-on'*:

assumes *finite* R

shows *finite* $\{(l, r, u).$

$\exists h\ us. u = Fun\ h\ us \wedge (l, r) \in R \wedge r \supseteq u \wedge (h, length\ us) \in F \wedge \neg (l \triangleright u)\}$
 $\langle proof \rangle$

lemma *card-image-le'*:

assumes *finite* S

shows $card\ (\bigcup y \in S. \{x. x = f\ y\}) \leq card\ S$

$\langle proof \rangle$

lemma *subteq-of-map-imp-map*: $map-funs-term\ g\ s \supseteq t \implies \exists u. t = map-funs-term\ g\ u$
 $\langle proof \rangle$

lemma *map-funs-term-inj*:

assumes *inj* $(fg :: ('f \Rightarrow 'g))$

shows *inj* $(map-funs-term\ fg)$

$\langle proof \rangle$

lemma *rsteps-closed-ctxt*:

assumes $(s, t) \in (rstep\ R)^*$

shows $(C\langle s \rangle, C\langle t \rangle) \in (rstep\ R)^*$

$\langle proof \rangle$

lemma *not-Nil-imp-last*: $xs \neq [] \implies \exists ys\ y. xs = ys@[y]$

$\langle proof \rangle$

lemma *Nil-or-last*: $xs = [] \vee (\exists ys\ y. xs = ys@[y])$

$\langle proof \rangle$

lemma *one-imp-ctxt-closed*: **assumes** *one*: $\bigwedge f\ bef\ s\ t\ aft. (s, t) \in r \implies (Fun\ f\ (bef\ @\ s\ \# \ aft), Fun\ f\ (bef\ @\ t\ \# \ aft)) \in r$

shows *ctxt.closed* r

$\langle proof \rangle$

lemma *ctxt-closed-nrrstep* [intro]: *ctxt.closed* $(nrrstep\ R)$

$\langle proof \rangle$

definition *all-ctxt-closed* :: '*f sig* \Rightarrow ('*f*, '*v*) *trs* \Rightarrow bool **where**

all-ctxt-closed *F r* $\iff (\forall f\ ts\ ss. (f, \text{length } ss) \in F \longrightarrow \text{length } ts = \text{length } ss \longrightarrow$
 $(\forall i. i < \text{length } ts \longrightarrow (ts ! i, ss ! i) \in r) \longrightarrow (\forall i. i < \text{length } ts \longrightarrow \text{funas-term}$
 $(ts ! i) \cup \text{funas-term } (ss ! i) \subseteq F) \longrightarrow (\text{Fun } f\ ts, \text{Fun } f\ ss) \in r) \wedge (\forall x. (\text{Var } x,$
 $\text{Var } x) \in r)$

lemma *all-ctxt-closedD*: *all-ctxt-closed* *F r* $\implies (f, \text{length } ss) \in F \implies \text{length } ts =$
 $\text{length } ss$

$\implies [\bigwedge i. i < \text{length } ts \implies (ts ! i, ss ! i) \in r]$
 $\implies [\bigwedge i. i < \text{length } ts \implies \text{funas-term } (ts ! i) \subseteq F]$
 $\implies [\bigwedge i. i < \text{length } ts \implies \text{funas-term } (ss ! i) \subseteq F]$
 $\implies (\text{Fun } f\ ts, \text{Fun } f\ ss) \in r$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-sig-reflE*: **assumes** *all*: *all-ctxt-closed* *F r*

shows *funas-term* *t* $\subseteq F \implies (t, t) \in r$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-reflE*: **assumes** *all*: *all-ctxt-closed* *UNIV r*

shows $(t, t) \in r$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-relcomp*: **assumes** *all-ctxt-closed* *UNIV R* *all-ctxt-closed* *UNIV S*

shows *all-ctxt-closed* *UNIV (R O S)*
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-relpow*:

assumes *acc*: *all-ctxt-closed* *UNIV Q*
shows *all-ctxt-closed* *UNIV (Q \rightsquigarrow^n)*
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-subst-step-sig*:

fixes *r* :: ('*f*, '*v*) *trs* **and** *t* :: ('*f*, '*v*) *term*
assumes *all*: *all-ctxt-closed* *F r*
and *sig*: *funas-term* *t* $\subseteq F$
and *steps*: $\bigwedge x. x \in \text{vars-term } t \implies (\sigma\ x, \tau\ x) \in r$
and *sig-subst*: $\bigwedge x. x \in \text{vars-term } t \implies \text{funas-term } (\sigma\ x) \cup \text{funas-term } (\tau\ x)$
 $\subseteq F$
shows $(t \cdot \sigma, t \cdot \tau) \in r$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-subst-step*:

fixes *r* :: ('*f*, '*v*) *trs* **and** *t* :: ('*f*, '*v*) *term*
assumes *all*: *all-ctxt-closed* *UNIV r*
and *steps*: $\bigwedge x. x \in \text{vars-term } t \implies (\sigma\ x, \tau\ x) \in r$
shows $(t \cdot \sigma, t \cdot \tau) \in r$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-ctxtE*: **assumes** *all*: *all-ctxt-closed* F R

and Fs : *funas-term* $s \subseteq F$

and Ft : *funas-term* $t \subseteq F$

and *step*: $(s, t) \in R$

shows *funas-ctxt* $C \subseteq F \implies (C \langle s \rangle, C \langle t \rangle) \in R$

<proof>

lemma *trans-ctxt-sig-imp-all-ctxt-closed*: **assumes** *tran*: *trans* r

and *refl*: $\bigwedge t. \text{funas-term } t \subseteq F \implies (t, t) \in r$

and *ctxt*: $\bigwedge C s t. \text{funas-ctxt } C \subseteq F \implies \text{funas-term } s \subseteq F \implies \text{funas-term } t \subseteq F \implies (s, t) \in r \implies (C \langle s \rangle, C \langle t \rangle) \in r$

shows *all-ctxt-closed* F r

<proof>

lemma *trans-ctxt-imp-all-ctxt-closed*: **assumes** *tran*: *trans* r

and *refl*: *refl* r

and *ctxt*: *ctxt.closed* r

shows *all-ctxt-closed* F r

<proof>

lemma *all-ctxt-closed-rsteps[intro]*: *all-ctxt-closed* F $((\text{rstep } r)^*)$

<proof>

lemma *subst-rsteps-imp-rsteps*:

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

assumes $\bigwedge x. x \in \text{vars-term } t \implies (\sigma x, \tau x) \in (\text{rstep } R)^*$

shows $(t \cdot \sigma, t \cdot \tau) \in (\text{rstep } R)^*$

<proof>

lemma *rtrancl-trancl-into-trancl*:

assumes *len*: *length* $ts = \text{length } ss$

and *steps*: $\forall i < \text{length } ts. (ts ! i, ss ! i) \in R^*$

and *i*: $i < \text{length } ts$

and *step*: $(ts ! i, ss ! i) \in R^+$

and *ctxt*: *ctxt.closed* R

shows $(\text{Fun } f \text{ } ts, \text{Fun } f \text{ } ss) \in R^+$

<proof>

lemma *SN-ctxt-apply-imp-SN-ctxt-to-term-list-gen*:

assumes *ctxt*: *ctxt.closed* r

assumes *SN*: *SN-on* $r \{C(t)\}$

shows *SN-on* $r (\text{set } (\text{ctxt-to-term-list } C))$

<proof>

lemma *rstep-subset*: *ctxt.closed* $R' \implies \text{subst.closed } R' \implies R \subseteq R' \implies \text{rstep } R \subseteq R' \text{ } \langle \text{proof} \rangle$

lemma *trancl-rstep-ctxt*:

$(s, t) \in (rstep\ R)^+ \implies (C\langle s \rangle, C\langle t \rangle) \in (rstep\ R)^+$
 $\langle proof \rangle$

lemma *args-steps-imp-steps-gen*:

assumes *ctxt*: $\bigwedge\ bef\ s\ t\ aft. (s, t) \in r\ (length\ bef) \implies$
 $length\ ts = Suc\ (length\ bef + length\ aft) \implies$
 $(Fun\ f\ (bef\ @\ (s :: ('f, 'v)\ term)\ \# \ aft), Fun\ f\ (bef\ @\ t\ \# \ aft)) \in R^*$
and *len*: $length\ ss = length\ ts$
and *args*: $\bigwedge\ i. i < length\ ts \implies (ss!\ i, ts!\ i) \in (r\ i)^*$
shows $(Fun\ f\ ss, Fun\ f\ ts) \in R^*$
 $\langle proof \rangle$

lemma *args-steps-imp-steps*:

assumes *ctxt*: *ctxt.closed* *R*
and *len*: $length\ ss = length\ ts$ **and** *args*: $\forall i < length\ ss. (ss!\ i, ts!\ i) \in R^*$
shows $(Fun\ f\ ss, Fun\ f\ ts) \in R^*$
 $\langle proof \rangle$

lemmas *args-rsteps-imp-rsteps* = *args-steps-imp-steps* [*OF* *ctxt-closed-rstep*]

lemma *replace-at-subst-steps*:

fixes $\sigma\ \tau :: ('f, 'v)\ subst$
assumes *acc*: *all-ctxt-closed* *UNIV* *r*
and *refl*: *refl* *r*
and $*$: $\bigwedge x. (\sigma\ x, \tau\ x) \in r$
and *p* $\in poss\ t$
and $t \mid -\ p = Var\ x$
shows $(replace\ at\ (t \cdot \sigma)\ p\ (\tau\ x), t \cdot \tau) \in r$
 $\langle proof \rangle$

lemma *replace-at-subst-rsteps*:

fixes $\sigma\ \tau :: ('f, 'v)\ subst$
assumes $*$: $\bigwedge x. (\sigma\ x, \tau\ x) \in (rstep\ R)^*$
and *p* $\in poss\ t$
and $t \mid -\ p = Var\ x$
shows $(replace\ at\ (t \cdot \sigma)\ p\ (\tau\ x), t \cdot \tau) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *substs-rsteps*:

assumes $\bigwedge x. (\sigma\ x, \tau\ x) \in (rstep\ R)^*$
shows $(t \cdot \sigma, t \cdot \tau) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *nrrstep-Fun-imp-arg-rstep*:

fixes *ss* :: $('f, 'v)\ term\ list$
assumes $(Fun\ f\ ss, Fun\ f\ ts) \in nrrstep\ R$ (**is** $(?s, ?t) \in nrrstep\ R$)
shows $\exists C\ i. i < length\ ss \wedge (ss!\ i, ts!\ i) \in rstep\ R \wedge C\langle ss!\ i \rangle = Fun\ f\ ss \wedge C\langle ts!\ i \rangle$
 $= Fun\ f\ ts$
 $\langle proof \rangle$

lemma *pair-fun-eq[simp]*:
fixes $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$
shows $((\lambda(x,y). (x,f\ y)) \circ (\lambda(x,y). (x,g\ y))) = (\lambda(x,y). (x,(f \circ g)\ y))$ (**is** $?f = ?g$)
 $\langle proof \rangle$

lemma *restrict-singleton*:
assumes $x \in \text{subst-domain } \sigma$ **shows** $\exists t. \sigma \mid s\ \{x\} = (\lambda y. \text{if } y = x \text{ then } t \text{ else } \text{Var } y)$
 $\langle proof \rangle$

definition *rstep-r-c-s* $:: ('f, 'v)\text{rule} \Rightarrow ('f, 'v)\text{ctxt} \Rightarrow ('f, 'v)\text{subst} \Rightarrow ('f, 'v)\text{term rel}$
where $\text{rstep-r-c-s } r\ C\ \sigma = \{(s,t) \mid s\ t. s = C\langle \text{fst } r \cdot \sigma \rangle \wedge t = C\langle \text{snd } r \cdot \sigma \rangle\}$

lemma *rstep-iff-rstep-r-c-s*: $((s,t) \in \text{rstep } R) = (\exists\ l\ r\ C\ \sigma. (l,r) \in R \wedge (s,t) \in \text{rstep-r-c-s } (l,r)\ C\ \sigma)$ (**is** $?left = ?right$)
 $\langle proof \rangle$

lemma *rstep-subset-characterization*:
 $(\text{rstep } R \subseteq \text{rstep } S) = (\forall\ l\ r. (l,r) \in R \longrightarrow (\exists\ l'\ r'\ C\ \sigma. (l',r') \in S \wedge l = C\langle l' \cdot \sigma \rangle \wedge r = C\langle r' \cdot \sigma \rangle))$ (**is** $?left = ?right$)
 $\langle proof \rangle$

lemma *rstep-preserves-funas-terms-var-cond*:
assumes $\text{funas-trs } R \subseteq F$ **and** $\text{funas-term } s \subseteq F$ **and** $(s,t) \in \text{rstep } R$
and $\text{wf}: \bigwedge l\ r. (l,r) \in R \implies \text{vars-term } r \subseteq \text{vars-term } l$
shows $\text{funas-term } t \subseteq F$
 $\langle proof \rangle$

lemma *rstep-preserves-funas-terms*:
assumes $\text{funas-trs } R \subseteq F$ **and** $\text{funas-term } s \subseteq F$ **and** $(s,t) \in \text{rstep } R$
and $\text{wf}: \text{wf-trs } R$
shows $\text{funas-term } t \subseteq F$
 $\langle proof \rangle$

lemma *rsteps-preserve-funas-terms-var-cond*:
assumes $F: \text{funas-trs } R \subseteq F$ **and** $s: \text{funas-term } s \subseteq F$ **and** $\text{steps}: (s,t) \in (\text{rstep } R)^*$
and $\text{wf}: \bigwedge l\ r. (l,r) \in R \implies \text{vars-term } r \subseteq \text{vars-term } l$
shows $\text{funas-term } t \subseteq F$
 $\langle proof \rangle$

lemma *rsteps-preserve-funas-terms*:
assumes $F: \text{funas-trs } R \subseteq F$ **and** $s: \text{funas-term } s \subseteq F$
and $\text{steps}: (s,t) \in (\text{rstep } R)^*$ **and** $\text{wf}: \text{wf-trs } R$
shows $\text{funas-term } t \subseteq F$
 $\langle proof \rangle$

lemma *no-Var-rstep* [*simp*]:
 assumes *wf-trs* *R* and $(\text{Var } x, t) \in \text{rstep } R$ shows *False*
 ⟨*proof*⟩

lemma *lhs-wf*:
 assumes *R*: $(l, r) \in R$ and *funas-trs* $R \subseteq F$
 shows *funas-term* $l \subseteq F$
 ⟨*proof*⟩

lemma *rhs-wf*:
 assumes *R*: $(l, r) \in R$ and *funas-trs* $R \subseteq F$
 shows *funas-term* $r \subseteq F$
 ⟨*proof*⟩

lemma *supt-map-funs-term* [*intro*]:
 assumes $t \triangleright s$
 shows *map-funs-term* $fg\ t \triangleright \text{map-funs-term } fg\ s$
 ⟨*proof*⟩

lemma *nondef-root-imp-arg-step*:
 assumes $(\text{Fun } f\ ss, t) \in \text{rstep } R$
 and *wf*: $\forall (l, r) \in R. \text{is-Fun } l$
 and *ndef*: $\neg \text{defined } R\ (f, \text{length } ss)$
 shows $\exists i < \text{length } ss. (ss\ !\ i, t\ |- [i]) \in \text{rstep } R$
 $\wedge t = \text{Fun } f\ (\text{take } i\ ss\ @\ (t\ |- [i])\ \# \text{drop } (\text{Suc } i)\ ss)$
 ⟨*proof*⟩

lemma *nondef-root-imp-arg-steps*:
 assumes $(\text{Fun } f\ ss, t) \in (\text{rstep } R)^*$
 and *wf*: $\forall (l, r) \in R. \text{is-Fun } l$
 and $\neg \text{defined } R\ (f, \text{length } ss)$
 shows $\exists ts. \text{length } ts = \text{length } ss \wedge t = \text{Fun } f\ ts \wedge (\forall i < \text{length } ss. (ss\ !\ i, ts\ !\ i) \in (\text{rstep } R)^*)$
 ⟨*proof*⟩

lemma *rstep-imp-nrrstep*:
 assumes *is-Fun* *s* and $\neg \text{defined } R\ (\text{the } (\text{root } s))$ and $\forall (l, r) \in R. \text{is-Fun } l$
 and $(s, t) \in \text{rstep } R$
 shows $(s, t) \in \text{nrrstep } R$
 ⟨*proof*⟩

lemma *rsteps-imp-nrrsteps*:
 assumes *is-Fun* *s* and $\neg \text{defined } R\ (\text{the } (\text{root } s))$
 and *no-vars*: $\forall (l, r) \in R. \text{is-Fun } l$
 and $(s, t) \in (\text{rstep } R)^*$
 shows $(s, t) \in (\text{nrrstep } R)^*$
 ⟨*proof*⟩

lemma *left-var-imp-not-SN*:
fixes $R :: ('f, 'v)trs$ **and** $t :: ('f, 'v) term$
assumes $(Var\ y, r) \in R$ **(is** $(?y, -) \in -$)
shows $\neg (SN\text{-on}\ (rstep\ R)\ \{t\})$
 $\langle proof \rangle$

lemma *not-SN-subt-imp-not-SN*:
assumes $ctxt: ctxt.closed\ R$ **and** $SN: \neg SN\text{-on}\ R\ \{t\}$ **and** $sub: s \sqsupseteq t$
shows $\neg SN\text{-on}\ R\ \{s\}$
 $\langle proof \rangle$

lemma *root-Some*:
assumes $root\ t = Some\ fn$
obtains ss **where** $length\ ss = snd\ fn$ **and** $t = Fun\ (fst\ fn)\ ss$
 $\langle proof \rangle$

lemma *map-funs-rule-power*:
fixes $f :: 'f \Rightarrow 'f$
shows $((map\text{-funs-rule}\ f) \text{~}^n) = map\text{-funs-rule}\ (f \text{~}^n)$
 $\langle proof \rangle$

lemma *map-funs-trs-power*:
fixes $f :: 'f \Rightarrow 'f$
shows $map\text{-funs-trs}\ f \text{~}^n = map\text{-funs-trs}\ (f \text{~}^n)$
 $\langle proof \rangle$

The set of minimally nonterminating terms with respect to a relation R .

definition $Tinf :: ('f, 'v) trs \Rightarrow ('f, 'v) terms$
where
 $Tinf\ R = \{t. \neg SN\text{-on}\ R\ \{t\} \wedge (\forall s \triangleleft t. SN\text{-on}\ R\ \{s\})\}$

lemma *not-SN-imp-subt-Tinf*:
assumes $\neg SN\text{-on}\ R\ \{s\}$ **shows** $\exists t \trianglelefteq s. t \in Tinf\ R$
 $\langle proof \rangle$

lemma *not-SN-imp-Tinf*:
assumes $\neg SN\ R$ **shows** $\exists t. t \in Tinf\ R$
 $\langle proof \rangle$

lemma *ctxt-of-pos-term-map-funs-term-conv [iff]*:
assumes $p \in poss\ s$
shows $map\text{-funs-ctxt}\ fg\ (ctxt\text{-of-pos-term}\ p\ s) = (ctxt\text{-of-pos-term}\ p\ (map\text{-funs-term}\ fg\ s))$
 $\langle proof \rangle$

lemma *var-rewrite-imp-not-SN*:
assumes $sn: SN\text{-on}\ (rstep\ R)\ \{u\}$ **and** $step: (t, s) \in rstep\ R$
shows $is\text{-Fun}\ t$
 $\langle proof \rangle$

lemma *rstep-id*: $rstep\ Id = Id$ $\langle proof \rangle$

lemma *map-funs-rule-id* [*simp*]: $map-funs-rule\ id = id$
 $\langle proof \rangle$

lemma *map-funs-trs-id* [*simp*]: $map-funs-trs\ id = id$
 $\langle proof \rangle$

definition *sig-step* :: $'f\ sig \Rightarrow ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs$ **where**
 $sig-step\ F\ R = \{(a, b). (a, b) \in R \wedge funas-term\ a \subseteq F \wedge funas-term\ b \subseteq F\}$

lemma *sig-step-union*: $sig-step\ F\ (R \cup S) = sig-step\ F\ R \cup sig-step\ F\ S$
 $\langle proof \rangle$

lemma *sig-step-UNIV*: $sig-step\ UNIV\ R = R$ $\langle proof \rangle$

lemma *sig-stepI*[*intro*]: $(a, b) \in R \Longrightarrow funas-term\ a \subseteq F \Longrightarrow funas-term\ b \subseteq F$
 $\Longrightarrow (a, b) \in sig-step\ F\ R$ $\langle proof \rangle$

lemma *sig-stepE*[*elim, consumes 1*]: $(a, b) \in sig-step\ F\ R \Longrightarrow \llbracket (a, b) \in R \Longrightarrow funas-term\ a \subseteq F \Longrightarrow funas-term\ b \subseteq F \Longrightarrow P \rrbracket \Longrightarrow P$ $\langle proof \rangle$

lemma *all-ctxt-closed-sig-rsteps* [*intro*]:
fixes $R :: ('f, 'v)\ trs$
shows $all-ctxt-closed\ F\ ((sig-step\ F\ (rstep\ R))^*)$ (**is** $all-ctxt-closed - (?R^*)$)
 $\langle proof \rangle$

lemma *wf-loop-imp-sig-ctxt-rel-not-SN*:
assumes $R: (l, C\langle l \rangle) \in R$ **and** $wf-l: funas-term\ l \subseteq F$
and $wf-C: funas-ctxt\ C \subseteq F$
and $ctxt: ctxt.closed\ R$
shows $\neg SN-on\ (sig-step\ F\ R)\ \{l\}$
 $\langle proof \rangle$

lemma *lhs-var-imp-sig-step-not-SN-on*:
assumes $x: (Var\ x, r) \in R$ **and** $F: funas-trs\ R \subseteq F$
shows $\neg SN-on\ (sig-step\ F\ (rstep\ R))\ \{Var\ x\}$
 $\langle proof \rangle$

lemma *rhs-free-vars-imp-sig-step-not-SN*:
assumes $R: (l, r) \in R$ **and** $free: \neg vars-term\ r \subseteq vars-term\ l$
and $F: funas-trs\ R \subseteq F$
shows $\neg SN-on\ (sig-step\ F\ (rstep\ R))\ \{l\}$
 $\langle proof \rangle$

lemma *lhs-var-imp-rstep-not-SN*: **assumes** $(Var\ x, r) \in R$ **shows** $\neg SN(rstep\ R)$
 $\langle proof \rangle$

lemma *rhs-free-vars-imp-rstep-not-SN*:

assumes $(l,r) \in R$ **and** $\neg \text{vars-term } r \subseteq \text{vars-term } l$

shows $\neg \text{SN-on } (\text{rstep } R) \{l\}$

$\langle \text{proof} \rangle$

lemma *free-right-rewrite-imp-not-SN*:

assumes $\text{step}: (t,s) \in \text{rstep-r-p-s } R (l,r) \text{ p } \sigma$

and $\text{vars}: \neg \text{vars-term } l \supseteq \text{vars-term } r$

shows $\neg \text{SN-on } (\text{rstep } R) \{t\}$

$\langle \text{proof} \rangle$

lemma *not-SN-on-rstep-subst-apply-term[intro]*:

assumes $\neg \text{SN-on } (\text{rstep } R) \{t\}$ **shows** $\neg \text{SN-on } (\text{rstep } R) \{t \cdot \sigma\}$

$\langle \text{proof} \rangle$

lemma *SN-rstep-imp-wf-trs*: **assumes** $\text{SN } (\text{rstep } R)$ **shows** $\text{wf-trs } R$

$\langle \text{proof} \rangle$

lemma *SN-sig-step-imp-wf-trs*: **assumes** $\text{SN}: \text{SN } (\text{sig-step } F (\text{rstep } R))$ **and** $F: \text{funas-trs } R \subseteq F$ **shows** $\text{wf-trs } R$

$\langle \text{proof} \rangle$

lemma *rstep-cases'[consumes 1, case-names root nonroot]*:

assumes $\text{rstep}: (s, t) \in \text{rstep } R$

and $\text{root}: \bigwedge l \text{ r } \sigma. (l, r) \in R \implies l \cdot \sigma = s \implies r \cdot \sigma = t \implies P$

and $\text{nonroot}: \bigwedge f \text{ ss1 } u \text{ ss2 } v. s = \text{Fun } f (\text{ss1 } @ u \# \text{ss2}) \implies t = \text{Fun } f (\text{ss1 } @ v \# \text{ss2}) \implies (u, v) \in \text{rstep } R \implies P$

shows P

$\langle \text{proof} \rangle$

lemma *NF-Var*: **assumes** $\text{wf}: \text{wf-trs } R$ **shows** $(\text{Var } x, t) \notin \text{rstep } R$

$\langle \text{proof} \rangle$

lemma *rstep-cases-Fun'[consumes 2, case-names root nonroot]*:

assumes $\text{wf}: \text{wf-trs } R$

and $\text{rstep}: (\text{Fun } f \text{ ss}, t) \in \text{rstep } R$

and $\text{root}': \bigwedge ls \text{ r } \sigma. (\text{Fun } f \text{ ls}, r) \in R \implies \text{map } (\lambda t. t \cdot \sigma) \text{ ls} = \text{ss} \implies r \cdot \sigma = t \implies P$

and $\text{nonroot}': \bigwedge i \text{ u}. i < \text{length } \text{ss} \implies t = \text{Fun } f (\text{take } i \text{ ss} @ u \# \text{drop } (\text{Suc } i) \text{ ss}) \implies (\text{ss}!i, u) \in \text{rstep } R \implies P$

shows P

$\langle \text{proof} \rangle$

lemma *rstep-preserves-undefined-root*:

assumes $\text{wf-trs } R$ **and** $\neg \text{defined } R (f, \text{length } \text{ss})$ **and** $(\text{Fun } f \text{ ss}, t) \in \text{rstep } R$

shows $\exists ts. \text{length } ts = \text{length } \text{ss} \wedge t = \text{Fun } f \text{ ts}$

$\langle \text{proof} \rangle$

lemma *rstep-ctxt-imp-nrrstep*: **assumes** $\text{step}: (s,t) \in \text{rstep } R$ **and** $C: C \neq \square$

shows $(C\langle s \rangle, C\langle t \rangle) \in \text{nrrstep } R$
 $\langle \text{proof} \rangle$

lemma *rstps-ctxt-imp-nrrsteps*: **assumes** $\text{steps}: (s, t) \in (\text{rstep } R)^*$ **and** $C: C \neq \square$
shows $(C\langle s \rangle, C\langle t \rangle) \in (\text{nrrstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *nrrstep-mono*:
assumes $R \subseteq R'$
shows $\text{nrrstep } R \subseteq \text{nrrstep } R'$
 $\langle \text{proof} \rangle$

lemma *rrstepE*:
assumes $(s, t) \in \text{rrstep } R$
obtains l **and** r **and** σ **where** $(l, r) \in R$ **and** $s = l \cdot \sigma$ **and** $t = r \cdot \sigma$
 $\langle \text{proof} \rangle$

lemma *nrrstepE*:
assumes $(s, t) \in \text{nrrstep } R$
obtains C **and** l **and** r **and** σ **where** $C \neq \square$ **and** $(l, r) \in R$
and $s = C\langle l \cdot \sigma \rangle$ **and** $t = C\langle r \cdot \sigma \rangle$
 $\langle \text{proof} \rangle$

lemma *singleton-subst-restrict* [*simp*]:
 $\text{subst } x \ s \mid s \ \{x\} = \text{subst } x \ s$
 $\langle \text{proof} \rangle$

lemma *singleton-subst-map* [*simp*]:
 $f \circ \text{subst } x \ s = (f \circ \text{Var})(x := f \ s) \ \langle \text{proof} \rangle$

lemma *subst-restrict-vars* [*simp*]:
 $(\lambda z. \text{if } z \in V \text{ then } f \ z \text{ else } g \ z) \mid s \ V = f \mid s \ V$
 $\langle \text{proof} \rangle$

lemma *subst-restrict-restrict* [*simp*]:
assumes $V \cap W = \{\}$
shows $((\lambda z. \text{if } z \in V \text{ then } f \ z \text{ else } g \ z) \mid s \ W) = g \mid s \ W$
 $\langle \text{proof} \rangle$

lemma *rstep-rstep*: $\text{rstep } (\text{rstep } R) = \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma *rstep-trancl-distrib*: $\text{rstep } (R^+) \subseteq (\text{rstep } R)^+$
 $\langle \text{proof} \rangle$

lemma *rstps-closed-subst*:
assumes $(s, t) \in (\text{rstep } R)^*$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\text{rstep } R)^*$

$\langle \text{proof} \rangle$

lemma *join-subst*:

subst.closed $r \implies (s, t) \in r^\downarrow \implies (s \cdot \sigma, t \cdot \sigma) \in r^\downarrow$
 $\langle \text{proof} \rangle$

lemma *join-subst-rstep* [intro]:

$(s, t) \in (\text{rstep } R)^\downarrow \implies (s \cdot \sigma, t \cdot \sigma) \in (\text{rstep } R)^\downarrow$
 $\langle \text{proof} \rangle$

lemma *join-ctxt* [intro]:

assumes $(s, t) \in (\text{rstep } R)^\downarrow$
shows $(C\langle s \rangle, C\langle t \rangle) \in (\text{rstep } R)^\downarrow$
 $\langle \text{proof} \rangle$

lemma *rstep-simps*:

$\text{rstep } (R^=) = (\text{rstep } R)^=$
 $\text{rstep } (\text{rstep } R) = \text{rstep } R$
 $\text{rstep } (R \cup S) = \text{rstep } R \cup \text{rstep } S$
 $\text{rstep } \text{Id} = \text{Id}$
 $\text{rstep } (R^{\leftrightarrow}) = (\text{rstep } R)^{\leftrightarrow}$
 $\langle \text{proof} \rangle$

lemma *rstep-rtrancl-idemp* [simp]:

$\text{rstep } ((\text{rstep } R)^*) = (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-rstep-conversion*:

all-ctxt-closed UNIV $((\text{rstep } R)^{\leftrightarrow*})$
 $\langle \text{proof} \rangle$

definition *instance-rule* :: $(f, v) \text{ rule} \Rightarrow (f, w) \text{ rule} \Rightarrow \text{bool}$ **where**

[code del]: *instance-rule* $lr \ st \longleftrightarrow (\exists \sigma. \text{fst } lr = \text{fst } st \cdot \sigma \wedge \text{snd } lr = \text{snd } st \cdot \sigma)$

definition *eq-rule-mod-vars* :: $(f, v) \text{ rule} \Rightarrow (f, v) \text{ rule} \Rightarrow \text{bool}$ **where**

eq-rule-mod-vars $lr \ st \longleftrightarrow \text{instance-rule } lr \ st \wedge \text{instance-rule } st \ lr$

notation *eq-rule-mod-vars* $((-/ =_v -) [51, 51] 50)$

lemma *instance-rule-var-cond*: **assumes** *eq*: *instance-rule* $(s, t) \ (l, r)$

and *vars*: *vars-term* $r \subseteq \text{vars-term } l$

shows *vars-term* $t \subseteq \text{vars-term } s$

$\langle \text{proof} \rangle$

lemma *instance-rule-rstep*: **assumes** *step*: $(s, t) \in \text{rstep } \{lr\}$

and *bex*: *Bex* $R \ (\text{instance-rule } lr)$

shows $(s, t) \in rstep\ R$
 $\langle proof \rangle$

lemma *eq-rule-mod-vars-var-cond*: **assumes** $eq: (l, r) =_v (s, t)$
and $vars: vars-term\ r \subseteq vars-term\ l$
shows $vars-term\ t \subseteq vars-term\ s$
 $\langle proof \rangle$

lemma *eq-rule-mod-varsE[elim]*: **fixes** $l :: ('f, 'v)term$
assumes $(l, r) =_v (s, t)$
shows $\exists\ \sigma\ \tau. l = s \cdot \sigma \wedge r = t \cdot \sigma \wedge s = l \cdot \tau \wedge t = r \cdot \tau \wedge range\ \sigma \subseteq range\ Var \wedge range\ \tau \subseteq range\ Var$
 $\langle proof \rangle$

definition

$linear-trs :: ('f, 'v)\ trs \Rightarrow bool$

where

$linear-trs\ R \equiv \forall (l, r) \in R. linear-term\ l \wedge linear-term\ r$

lemma *linear-trsE[elim, consumes 1]*: $linear-trs\ R \Longrightarrow (l, r) \in R \Longrightarrow linear-term\ l \wedge linear-term\ r$
 $\langle proof \rangle$

lemma *linear-trsI[intro]*: $\llbracket \bigwedge l\ r. (l, r) \in R \Longrightarrow linear-term\ l \wedge linear-term\ r \rrbracket \Longrightarrow linear-trs\ R$
 $\langle proof \rangle$

definition

$left-linear-trs :: ('f, 'v)\ trs \Rightarrow bool$

where

$left-linear-trs\ R \longleftrightarrow (\forall (l, r) \in R. linear-term\ l)$

lemma *left-linear-trs-union*: $left-linear-trs\ (R \cup S) = (left-linear-trs\ R \wedge left-linear-trs\ S)$
 $\langle proof \rangle$

lemma *left-linear-mono*: **assumes** $left-linear-trs\ S$ **and** $R \subseteq S$ **shows** $left-linear-trs\ R$
 $\langle proof \rangle$

lemma *left-linear-map-funs-trs[simp]*: $left-linear-trs\ (map-funs-trs\ f\ R) = left-linear-trs\ R$
 $\langle proof \rangle$

lemma *left-linear-weak-match-rstep*:
assumes $rstep: (u, v) \in rstep\ R$
and $weak-match: weak-match\ s\ u$
and $ll: left-linear-trs\ R$
shows $\exists t. (s, t) \in rstep\ R \wedge weak-match\ t\ v$

$\langle proof \rangle$

context
begin

private fun S **where**

$S\ R\ s\ t\ 0 = s$
 $| S\ R\ s\ t\ (Suc\ i) = (SOME\ u.\ (S\ R\ s\ t\ i, u) \in rstep\ R \wedge weak-match\ u\ (t(Suc\ i)))$

lemma *weak-match-SN*:

assumes wm : *weak-match* $s\ t$
and ll : *left-linear-trs* R
and SN : *SN-on* ($rstep\ R$) $\{s\}$
shows *SN-on* ($rstep\ R$) $\{t\}$

$\langle proof \rangle$

end

lemma *lhs-notin-NF-rstep*: $(l, r) \in R \implies l \notin NF\ (rstep\ R)$ $\langle proof \rangle$

lemma *NF-instance*:

assumes $(t \cdot \sigma) \in NF\ (rstep\ R)$ **shows** $t \in NF\ (rstep\ R)$
 $\langle proof \rangle$

lemma *NF-subterm*:

assumes $t \in NF\ (rstep\ R)$ **and** $t \supseteq s$
shows $s \in NF\ (rstep\ R)$
 $\langle proof \rangle$

abbreviation

$lhss :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ terms$

where

$lhss\ R \equiv fst\ ' R$

abbreviation

$rhss :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ terms$

where

$rhss\ R \equiv snd\ ' R$

definition *map-funs-trs-wa* :: $('f \times nat \Rightarrow 'g) \Rightarrow ('f, 'v)\ trs \Rightarrow ('g, 'v)\ trs$ **where**
 $map-funs-trs-wa\ fg\ R = (\lambda(l, r). (map-funs-term-wa\ fg\ l, map-funs-term-wa\ fg\ r))\ ' R$

lemma *map-funs-trs-wa-union*: $map-funs-trs-wa\ fg\ (R \cup S) = map-funs-trs-wa\ fg\ R \cup map-funs-trs-wa\ fg\ S$
 $\langle proof \rangle$

lemma *map-funs-term-wa-compose*: $map-funs-term-wa\ gh\ (map-funs-term-wa\ fg\ t) = map-funs-term-wa\ (\lambda(f, n). gh\ (fg\ (f, n), n))\ t$

$\langle \text{proof} \rangle$

lemma *map-funs-trs-wa-compose*: *map-funs-trs-wa* *gh* (*map-funs-trs-wa* *fg* *R*) =
map-funs-trs-wa ($\lambda (f,n). \text{gh } (fg (f,n), n)$) *R* (**is** ?*L* = *map-funs-trs-wa* ?*fg* *R*)
 $\langle \text{proof} \rangle$

lemma *map-funs-trs-wa-funas-trs-id*: **assumes** *R*: *funas-trs* *R* $\subseteq F$
and *id*: $\bigwedge g n. (g,n) \in F \implies f (g,n) = g$
shows *map-funs-trs-wa* *f* *R* = *R*
 $\langle \text{proof} \rangle$

lemma *map-funs-trs-wa-rstep*: **assumes** *step*: $(s,t) \in \text{rstep } R$
shows (*map-funs-term-wa* *fg* *s*, *map-funs-term-wa* *fg* *t*) $\in \text{rstep } (\text{map-funs-trs-wa } fg \text{ } R)$
 $\langle \text{proof} \rangle$

lemma *map-funs-trs-wa-rsteps*: **assumes** *step*: $(s,t) \in (\text{rstep } R)^*$
shows (*map-funs-term-wa* *fg* *s*, *map-funs-term-wa* *fg* *t*) $\in (\text{rstep } (\text{map-funs-trs-wa } fg \text{ } R))^*$
 $\langle \text{proof} \rangle$

lemma *rstep-ground*:
assumes *wf-trs*: $\bigwedge l r. (l, r) \in R \implies \text{vars-term } r \subseteq \text{vars-term } l$
and *ground*: *ground* *s*
and *step*: $(s, t) \in \text{rstep } R$
shows *ground* *t*
 $\langle \text{proof} \rangle$

lemma *rsteps-ground*:
assumes *wf-trs*: $\bigwedge l r. (l, r) \in R \implies \text{vars-term } r \subseteq \text{vars-term } l$
and *ground*: *ground* *s*
and *steps*: $(s, t) \in (\text{rstep } R)^*$
shows *ground* *t*
 $\langle \text{proof} \rangle$

definition *locally-terminating* :: $(f,v) \text{trs} \Rightarrow \text{bool}$
where *locally-terminating* *R* $\equiv \forall F. \text{finite } F \longrightarrow \text{SN } (\text{sig-step } F (\text{rstep } R))$

lemma *supt-rstep-stable*:
assumes $(s, t) \in \{\triangleright\} \cup \text{rstep } R$
shows $(s \cdot \sigma, t \cdot \sigma) \in \{\triangleright\} \cup \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma *supt-rstep-trancl-stable*:
assumes $(s, t) \in (\{\triangleright\} \cup \text{rstep } R)^+$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\{\triangleright\} \cup \text{rstep } R)^+$
 $\langle \text{proof} \rangle$

lemma *supt-rsteps-stable*:

assumes $(s, t) \in (\{\triangleright\} \cup \text{rstep } R)^*$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\{\triangleright\} \cup \text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *eq-rule-mod-vars-refl[simp]*: $r =_v r$
 $\langle \text{proof} \rangle$

lemma *instance-rule-refl[simp]*: *instance-rule* r r
 $\langle \text{proof} \rangle$

lemma *is-Fun-Fun-conv*: *is-Fun* $t = (\exists f \text{ ts. } t = \text{Fun } f \text{ ts})$ $\langle \text{proof} \rangle$

lemma *wf-trs-def'*:
 $\text{wf-trs } R = (\forall (l, r) \in R. \text{is-Fun } l \wedge \text{vars-term } r \subseteq \text{vars-term } l)$
 $\langle \text{proof} \rangle$

definition *wf-rule* :: $(f, 'v) \text{ rule} \Rightarrow \text{bool}$ **where**
 $\text{wf-rule } r \longleftrightarrow \text{is-Fun } (\text{fst } r) \wedge \text{vars-term } (\text{snd } r) \subseteq \text{vars-term } (\text{fst } r)$

definition *wf-rules* :: $(f, 'v) \text{ trs} \Rightarrow (f, 'v) \text{ trs}$ **where**
 $\text{wf-rules } R = \{r. r \in R \wedge \text{wf-rule } r\}$

lemma *wf-trs-wf-rules[simp]*: *wf-trs* (*wf-rules* R)
 $\langle \text{proof} \rangle$

lemma *wf-rules-subset[simp]*: *wf-rules* $R \subseteq R$
 $\langle \text{proof} \rangle$

fun *wf-reltrs* :: $(f, 'v) \text{ trs} \Rightarrow (f, 'v) \text{ trs} \Rightarrow \text{bool}$ **where**
 $\text{wf-reltrs } R \text{ } S = (\text{wf-trs } R \wedge (R \neq \{\} \longrightarrow (\forall l \text{ r. } (l, r) \in S \longrightarrow \text{vars-term } r \subseteq \text{vars-term } l)))$

lemma *SN-rel-imp-wf-reltrs*:
assumes *SN-rel*: *SN-rel* (*rstep* R) (*rstep* S)
shows *wf-reltrs* R S
 $\langle \text{proof} \rangle$

lemmas *rstep-wf-rules-subset* = *rstep-mono*[*OF wf-rules-subset*]

definition *map-vars-trs* :: $(v \Rightarrow w) \Rightarrow (f, 'v) \text{ trs} \Rightarrow (f, 'w) \text{ trs}$ **where**
 $\text{map-vars-trs } f \text{ } R = (\lambda (l, r). (\text{map-vars-term } f \text{ } l, \text{map-vars-term } f \text{ } r)) \text{ ' } R$

lemma *map-vars-trs-rstep*:
assumes $(s, t) \in \text{rstep } (\text{map-vars-trs } f \text{ } R)$ (**is** - $\in \text{rstep } ?R$)
shows $(s \cdot \tau, t \cdot \tau) \in \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma *map-vars-rsteps*:
assumes $(s, t) \in (\text{rstep } (\text{map-vars-trs } f \text{ } R))^*$ (**is** - $\in (\text{rstep } ?R)^*$)

shows $(s \cdot \tau, t \cdot \tau) \in (rstep\ R)^*$
 $\langle proof \rangle$

lemma *rstep-subst-closed*: $(s, t) \in (rstep\ R)^+ \implies (s \cdot \sigma, t \cdot \sigma) \in (rstep\ R)^+$
 $\langle proof \rangle$

lemma *supteq-rtranc1-supt*:
 $(R^+ \ O \ \{\triangleright\}) \subseteq (\{\triangleright\} \cup R)^+ \text{ (is } ?l \subseteq ?r)$
 $\langle proof \rangle$

lemma *rrstepI[intro]*: $(l, r) \in R \implies s = l \cdot \sigma \implies t = r \cdot \sigma \implies (s, t) \in rrstep\ R$
 $\langle proof \rangle$

lemma *CS-rrstep-conv*: *subst.closure* = *rrstep*
 $\langle proof \rangle$

4.5 Rewrite steps at a fixed position

inductive-set *rstep-pos* :: $(f, v)\ trs \Rightarrow pos \Rightarrow (f, v)\ term\ rel$ **for** R **and** p
where

rule [intro]: $(l, r) \in R \implies p \in poss\ s \implies s \mid\!-\ p = l \cdot \sigma \implies$
 $(s, replace\text{-}at\ s\ p\ (r \cdot \sigma)) \in rstep\text{-}pos\ R\ p$

lemma *rstep-pos-subst*:
assumes $(s, t) \in rstep\text{-}pos\ R\ p$
shows $(s \cdot \sigma, t \cdot \sigma) \in rstep\text{-}pos\ R\ p$
 $\langle proof \rangle$

lemma *rstep-pos-rule*:
assumes $(l, r) \in R$
shows $(l, r) \in rstep\text{-}pos\ R\ []$
 $\langle proof \rangle$

lemma *rstep-pos-rstep-r-p-s-conv*:
 $rstep\text{-}pos\ R\ p = \{(s, t) \mid s\ t\ r\ \sigma. (s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ p\ \sigma\}$
 $\langle proof \rangle$

lemma *rstep-rstep-pos-conv*:
 $rstep\ R = \{(s, t) \mid s\ t\ p. (s, t) \in rstep\text{-}pos\ R\ p\}$
 $\langle proof \rangle$

lemma *rstep-pos-supt*:
assumes $(s, t) \in rstep\text{-}pos\ R\ p$
and $q: q \in poss\ u$ **and** $u: u \mid\!-\ q = s$
shows $(u, (ctx\text{-}of\text{-}pos\text{-}term\ q\ u)\langle t \rangle) \in rstep\text{-}pos\ R\ (q\ @\ p)$
 $\langle proof \rangle$

lemma *rrstep-rstep-pos-conv*:
 $rrstep\ R = rstep\text{-}pos\ R\ []$

$\langle proof \rangle$

lemma *rrstep-imp-rstep*:
assumes $(s, t) \in rrstep\ R$
shows $(s, t) \in rstep\ R$
 $\langle proof \rangle$

lemma *not-NF-rstep-imp-subteq-not-NF-rrstep*:
assumes $s \notin NF\ (rstep\ R)$
shows $\exists t \leq s. t \notin NF\ (rrstep\ R)$
 $\langle proof \rangle$

lemma *all-subt-NF-rrstep-iff-all-subt-NF-rstep*:
 $(\forall s \triangleleft t. s \in NF\ (rrstep\ R)) \longleftrightarrow (\forall s \triangleleft t. s \in NF\ (rstep\ R))$
 $\langle proof \rangle$

lemma *not-in-poss-imp-NF-rstep-pos [simp]*:
assumes $p \notin poss\ s$
shows $s \in NF\ (rstep-pos\ R\ p)$
 $\langle proof \rangle$

lemma *Var-rstep-imp-rstep-pos-Empty*:
assumes $(Var\ x, t) \in rstep\ R$
shows $(Var\ x, t) \in rstep-pos\ R\ []$
 $\langle proof \rangle$

lemma *rstep-args-NF-imp-rrstep*:
assumes $(s, t) \in rstep\ R$
and $\forall u \triangleleft s. u \in NF\ (rstep\ R)$
shows $(s, t) \in rrstep\ R$
 $\langle proof \rangle$

lemma *rstep-pos-imp-rstep-pos-Empty*:
assumes $(s, t) \in rstep-pos\ R\ p$
shows $(s \mid -\ p, t \mid -\ p) \in rstep-pos\ R\ []$
 $\langle proof \rangle$

lemma *rstep-pos-arg*:
assumes $(s, t) \in rstep-pos\ R\ p$
and $i < length\ ss$ **and** $ss\ !\ i = s$
shows $(Fun\ f\ ss, (ctxt-of-pos-term\ [i]\ (Fun\ f\ ss))\langle t \rangle) \in rstep-pos\ R\ (i \# p)$
 $\langle proof \rangle$

lemma *rstep-imp-max-pos*:
assumes $(s, t) \in rstep\ R$
shows $\exists u. \exists p \in poss\ s. (s, u) \in rstep-pos\ R\ p \wedge (\forall v \triangleleft s \mid -\ p. v \in NF\ (rstep\ R))$
 $\langle proof \rangle$

lemma *rhs-free-vars-imp-rstep-not-SN'*:

assumes $(l, r) \in R$ **and** $\neg \text{vars-term } r \subseteq \text{vars-term } l$
shows $\neg SN \text{ (rstep } R)$
 $\langle \text{proof} \rangle$

lemma *SN-imp-variable-condition*:
assumes $SN \text{ (rstep } R)$
shows $\forall (l, r) \in R. \text{vars-term } r \subseteq \text{vars-term } l$
 $\langle \text{proof} \rangle$

definition *non-collapsing* $R \longleftrightarrow (\forall lr \in R. \text{is-Fun (snd } lr))$

4.6 Parallel rewrite relation

the parallel rewrite relation

inductive-set *par-rstep* :: $(f, 'v)trs \Rightarrow (f, 'v)trs$ **for** $R :: (f, 'v)trs$
where *root-step*[*intro*]: $(s, t) \in R \Longrightarrow (s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep } R$
| *par-step-fun*[*intro*]: $\llbracket \bigwedge i. i < \text{length } ts \Longrightarrow (ss ! i, ts ! i) \in \text{par-rstep } R \rrbracket \Longrightarrow$
 $\text{length } ss = \text{length } ts$
 $\Longrightarrow (\text{Fun } f \text{ } ss, \text{Fun } f \text{ } ts) \in \text{par-rstep } R$
| *par-step-var*[*intro*]: $(\text{Var } x, \text{Var } x) \in \text{par-rstep } R$

lemma *par-rstep-refl*[*intro*]: $(t, t) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

lemma *all-ctxt-closed-par-rstep*[*intro*]: $\text{all-ctxt-closed } F \text{ (par-rstep } R)$
 $\langle \text{proof} \rangle$

lemma *args-par-rstep-pow-imp-par-rstep-pow*:
 $\text{length } xs = \text{length } ys \Longrightarrow \forall i < \text{length } xs. (xs ! i, ys ! i) \in \text{par-rstep } R \rightsquigarrow^n \Longrightarrow$
 $(\text{Fun } f \text{ } xs, \text{Fun } f \text{ } ys) \in \text{par-rstep } R \rightsquigarrow^n$
 $\langle \text{proof} \rangle$

lemma *ctxt-closed-par-rstep*[*intro*]: $\text{ctxt.closed (par-rstep } R)$
 $\langle \text{proof} \rangle$

lemma *subst-closed-par-rstep*: $(s, t) \in \text{par-rstep } R \Longrightarrow (s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

lemma *R-par-rstep*: $R \subseteq \text{par-rstep } R$
 $\langle \text{proof} \rangle$

lemma *par-rstep-rsteps*: $\text{par-rstep } R \subseteq (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *rstep-par-rstep*: $\text{rstep } R \subseteq \text{par-rstep } R$
 $\langle \text{proof} \rangle$

lemma *par-rsteps-rsteps*: $(\text{par-rstep } R)^* = (\text{rstep } R)^* \text{ (is ?P = ?R)}$

$\langle \text{proof} \rangle$

lemma *par-rsteps-union*: $(\text{par-rstep } A \cup \text{par-rstep } B)^* =$
 $(\text{rstep } (A \cup B))^*$
 $\langle \text{proof} \rangle$

lemma *par-rstep-inverse*: $\text{par-rstep } (R \hat{-} 1) = (\text{par-rstep } R) \hat{-} 1$
 $\langle \text{proof} \rangle$

lemma *par-rstep-conversion*: $(\text{rstep } R)^{\leftrightarrow*} = (\text{par-rstep } R)^{\leftrightarrow*}$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mono*: **assumes** $R \subseteq S$
shows $\text{par-rstep } R \subseteq \text{par-rstep } S$
 $\langle \text{proof} \rangle$

lemma *wf-trs-par-rstep*: **assumes** $\text{wf}: \bigwedge l \ r. (l, r) \in R \implies \text{is-Fun } l$
and step: $(\text{Var } x, t) \in \text{par-rstep } R$
shows $t = \text{Var } x$
 $\langle \text{proof} \rangle$

main lemma which tells us, that either a parallel rewrite step of $l \cdot \sigma$ is inside l , or we can do the step completely inside σ

lemma *par-rstep-linear-subst*: **assumes** $\text{lin}: \text{linear-term } l$
and step: $(l \cdot \sigma, t) \in \text{par-rstep } R$
shows $(\exists \tau. t = l \cdot \tau \wedge (\forall x \in \text{vars-term } l. (\sigma \ x, \tau \ x) \in \text{par-rstep } R) \vee$
 $(\exists C \ l'' \ l' \ r'. l = C \langle l'' \rangle \wedge \text{is-Fun } l'' \wedge (l', r') \in R \wedge (l'' \cdot \sigma = l' \cdot \tau) \wedge$
 $((C \cdot_c \sigma) \langle r' \cdot \tau \rangle, t) \in \text{par-rstep } R))$
 $\langle \text{proof} \rangle$

lemma *par-rstep-id*:
 $(s, t) \in R \implies (s, t) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

4.7 Function Symbols and Variables

4.8 Function Symbols and Variables

fun *root-list* :: $('f, 'v) \text{ term} \Rightarrow ('f \times \text{nat}) \text{ list}$
where
 $\text{root-list } (\text{Var } x) = []$
 $\text{root-list } (\text{Fun } f \ ts) = [(f, \text{length } ts)]$

definition *vars-rule-list* :: $('f, 'v) \text{ rule} \Rightarrow 'v \text{ list}$
where
 $\text{vars-rule-list } r = \text{vars-term-list } (\text{fst } r) @ \text{vars-term-list } (\text{snd } r)$

definition *funs-rule-list* :: $('f, 'v) \text{ rule} \Rightarrow 'f \text{ list}$
where
 $\text{funs-rule-list } r = \text{funs-term-list } (\text{fst } r) @ \text{funs-term-list } (\text{snd } r)$

definition *funas-rule-list* :: ('f, 'v) rule \Rightarrow ('f \times nat) list

where

funas-rule-list r = *funas-term-list* (fst r) @ *funas-term-list* (snd r)

definition *roots-rule-list* :: ('f, 'v) rule \Rightarrow ('f \times nat) list

where

roots-rule-list r = *root-list* (fst r) @ *root-list* (snd r)

definition *funas-args-rule-list* :: ('f, 'v) rule \Rightarrow ('f \times nat) list

where

funas-args-rule-list r = *funas-args-term-list* (fst r) @ *funas-args-term-list* (snd r)

lemma *set-vars-rule-list* [simp]:

set (*vars-rule-list* r) = *vars-rule* r

<proof>

lemma *set-funs-rule-list* [simp]:

set (*funs-rule-list* r) = *funs-rule* r

<proof>

lemma *set-funas-rule-list* [simp]:

set (*funas-rule-list* r) = *funas-rule* r

<proof>

lemma *set-roots-rule-list* [simp]:

set (*roots-rule-list* r) = *roots-rule* r

<proof>

lemma *set-funas-args-rule-list* [simp]:

set (*funas-args-rule-list* r) = *funas-args-rule* r

<proof>

definition *vars-trs-list* :: ('f, 'v) rule list \Rightarrow 'v list

where

vars-trs-list trs = *concat* (*map vars-rule-list* trs)

definition *funs-trs-list* :: ('f, 'v) rule list \Rightarrow 'f list

where

funs-trs-list trs = *concat* (*map funs-rule-list* trs)

definition *funas-trs-list* :: ('f, 'v) rule list \Rightarrow ('f \times nat) list

where

funas-trs-list trs = *concat* (*map funas-rule-list* trs)

definition *roots-trs-list* :: ('f, 'v) rule list \Rightarrow ('f \times nat) list

where

roots-trs-list trs = *remdups* (*concat* (*map roots-rule-list* trs))

definition *funas-args-trs-list* :: ('f, 'v) rule list \Rightarrow ('f \times nat) list
where

funas-args-trs-list trs = concat (map *funas-args-rule-list* trs)

lemma *set-vars-trs-list* [simp]:
 set (vars-trs-list trs) = vars-trs (set trs)
 <proof>

lemma *set-funs-trs-list* [simp]:
 set (funs-trs-list R) = funs-trs (set R)
 <proof>

lemma *set-funas-trs-list* [simp]:
 set (funas-trs-list R) = funas-trs (set R)
 <proof>

lemma *set-roots-trs-list* [simp]:
 set (roots-trs-list R) = roots-trs (set R)
 <proof>

lemma *set-funas-args-trs-list* [simp]:
 set (funas-args-trs-list R) = funas-args-trs (set R)
 <proof>

lemmas vars-list-defs = vars-trs-list-def vars-rule-list-def
lemmas funs-list-defs = funs-trs-list-def funs-rule-list-def
lemmas funas-list-defs = funas-trs-list-def funas-rule-list-def
lemmas roots-list-defs = roots-trs-list-def roots-rule-list-def
lemmas funas-args-list-defs = funas-args-trs-list-def funas-args-rule-list-def

lemma *vars-trs-list-Nil* [simp]:
 vars-trs-list [] = [] <proof>

context
 fixes R :: ('f, 'v) trs
 assumes wf-trs R
begin

lemma *funas-term-subst-rhs*:
 assumes funas-trs R \subseteq F and (l, r) \in R and funas-term (l \cdot σ) \subseteq F
 shows funas-term (r \cdot σ) \subseteq F
 <proof>

lemma *vars-rule-lhs*:
 r \in R \implies vars-rule r = vars-term (fst r)
 <proof>

end

abbreviation $NF\text{-}trs :: ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ terms}$ **where**
 $NF\text{-}trs \ R \equiv NF \ (rstep \ R)$

lemma $NF\text{-}trs\text{-}mono$: $r \subseteq s \implies NF\text{-}trs \ s \subseteq NF\text{-}trs \ r$
 $\langle proof \rangle$

lemma $NF\text{-}trs\text{-}union$: $NF\text{-}trs \ (R \cup S) = NF\text{-}trs \ R \cap NF\text{-}trs \ S$
 $\langle proof \rangle$

abbreviation $NF\text{-}terms :: ('f, 'v) \text{ terms} \Rightarrow ('f, 'v) \text{ terms}$ **where**
 $NF\text{-}terms \ Q \equiv NF \ (rstep \ (Id\text{-}on \ Q))$

lemma $NF\text{-}terms\text{-}anti\text{-}mono$:
 $Q \subseteq Q' \implies NF\text{-}terms \ Q' \subseteq NF\text{-}terms \ Q$
 $\langle proof \rangle$

lemma $lhs\text{-}var\text{-}not\text{-}NF$:
assumes $l \in T$ **and** $is\text{-}Var \ l$ **shows** $t \notin NF\text{-}terms \ T$
 $\langle proof \rangle$

lemma $not\text{-}NF\text{-}termsE[elim]$:
assumes $s \notin NF\text{-}terms \ Q$
obtains $l \ C \ \sigma$ **where** $l \in Q$ **and** $s = C\langle l \cdot \sigma \rangle$
 $\langle proof \rangle$

lemma $notin\text{-}NF\text{-}E \ [elim]$:
fixes $R :: ('f, 'v) \text{ trs}$
assumes $t \notin NF\text{-}trs \ R$
obtains $C \ l$ **and** $\sigma :: ('f, 'v) \text{ subst}$ **where** $l \in lhss \ R$ **and** $t = C\langle l \cdot \sigma \rangle$
 $\langle proof \rangle$

lemma $NF\text{-}ctxt\text{-}subst$: $NF\text{-}terms \ Q = \{t. \neg (\exists \ C \ q \ \sigma. t = C\langle q \cdot \sigma \rangle \wedge q \in Q)\}$ (**is**
 $- = ?R$)
 $\langle proof \rangle$

lemma $some\text{-}NF\text{-}imp\text{-}no\text{-}Var$:
assumes $t \in NF\text{-}terms \ Q$
shows $Var \ x \notin Q$
 $\langle proof \rangle$

lemma $NF\text{-}map\text{-}vars\text{-}term\text{-}inj$:
assumes inj : $\bigwedge x. n \ (m \ x) = x$ **and** NF : $t \in NF\text{-}terms \ Q$
shows $(map\text{-}vars\text{-}term \ m \ t) \in NF\text{-}terms \ (map\text{-}vars\text{-}term \ m \ ' Q)$
 $\langle proof \rangle$

lemma $notin\text{-}NF\text{-}terms$: $t \in Q \implies t \notin NF\text{-}terms \ Q$
 $\langle proof \rangle$

lemma $NF\text{-}termsI \ [intro]$:

assumes NF : $\bigwedge C l \sigma. t = C \langle l \cdot \sigma \rangle \implies l \in Q \implies False$
shows $t \in NF\text{-terms } Q$
 $\langle proof \rangle$

lemma $NF\text{-args-imp-}NF$:
assumes ss : $\bigwedge s. s \in set\ ss \implies s \in NF\text{-terms } Q$
and $someNF$: $t \in NF\text{-terms } Q$
and $root$: $Some\ (f, length\ ss) \notin root\ 'Q$
shows $(Fun\ f\ ss) \in NF\text{-terms } Q$
 $\langle proof \rangle$

lemma $NF\text{-Var-is-Fun}$:
assumes Q : $Ball\ Q\ is\ Fun$
shows $Var\ x \in NF\text{-terms } Q$
 $\langle proof \rangle$

lemma $NF\text{-terms-lhss } [simp]$: $NF\text{-terms } (lhss\ R) = NF\ (rstep\ R)$
 $\langle proof \rangle$

fun $fun\text{-poss-list} :: ('f, 'v)\ term \Rightarrow pos\ list$
where
 $fun\text{-poss-list } (Var\ x) = [] \mid$
 $fun\text{-poss-list } (Fun\ f\ ss) = ([] \# concat\ (map\ (\lambda\ (i, ps). map\ ((\#)\ i)\ ps)\ (zip\ [0..length\ ss]\ (map\ fun\text{-poss-list}\ ss))))$

lemma $set\text{-fun-poss-list } [simp]$:
 $set\ (fun\text{-poss-list } t) = fun\text{-poss } t$
 $\langle proof \rangle$

abbreviation $relstep\ R\ E \equiv relto\ (rstep\ R)\ (rstep\ E)$

lemma $args\text{-}SN\text{-on-relstep-nrrstep-imp-args-}SN\text{-on}$:
assumes SN : $\bigwedge u. s \triangleright u \implies SN\text{-on } (relstep\ R\ E)\ \{u\}$
and st : $(s, t) \in nrrstep\ (R \cup E)$
and $supt$: $t \triangleright u$
shows $SN\text{-on } (relstep\ R\ E)\ \{u\}$
 $\langle proof \rangle$

lemma $Tinf\text{-nrrstep}$:
assumes $tinf$: $s \in Tinf\ (relstep\ R\ E)$ **and** st : $(s, t) \in nrrstep\ (R \cup E)$
and t : $\neg SN\text{-on } (relstep\ R\ E)\ \{t\}$
shows $t \in Tinf\ (relstep\ R\ E)$
 $\langle proof \rangle$

lemma $subterm\text{-preserves-}SN\text{-on-relstep}$:
 $SN\text{-on } (relstep\ R\ E)\ \{s\} \implies s \supseteq t \implies SN\text{-on } (relstep\ R\ E)\ \{t\}$
 $\langle proof \rangle$

inductive-set $rstep\text{-rule} :: ('f, 'v)\ rule \Rightarrow ('f, 'v)\ term\ rel\ \mathbf{for}\ \varrho$

where

rule: $s = C\langle \text{fst } \varrho \cdot \sigma \rangle \implies t = C\langle \text{snd } \varrho \cdot \sigma \rangle \implies (s, t) \in \text{rstep-rule } \varrho$

lemma *rstep-ruleI* [intro]:

$s = C\langle l \cdot \sigma \rangle \implies t = C\langle r \cdot \sigma \rangle \implies (s, t) \in \text{rstep-rule } (l, r)$
 $\langle \text{proof} \rangle$

lemma *rstep-rule-ctxt*:

$(s, t) \in \text{rstep-rule } \varrho \implies (C\langle s \rangle, C\langle t \rangle) \in \text{rstep-rule } \varrho$
 $\langle \text{proof} \rangle$

lemma *rstep-rule-subst*:

assumes $(s, t) \in \text{rstep-rule } \varrho$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{rstep-rule } \varrho$
 $\langle \text{proof} \rangle$

lemma *rstep-rule-imp-rstep*:

$\varrho \in R \implies (s, t) \in \text{rstep-rule } \varrho \implies (s, t) \in \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma *rstep-imp-rstep-rule*:

assumes $(s, t) \in \text{rstep } R$
obtains $l\ r$ **where** $(l, r) \in R$ **and** $(s, t) \in \text{rstep-rule } (l, r)$
 $\langle \text{proof} \rangle$

lemma *term-subst-rstep*:

assumes $\bigwedge x. x \in \text{vars-term } t \implies (\sigma\ x, \tau\ x) \in \text{rstep } R$
shows $(t \cdot \sigma, t \cdot \tau) \in (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *term-subst-rsteps*:

assumes $\bigwedge x. x \in \text{vars-term } t \implies (\sigma\ x, \tau\ x) \in (\text{rstep } R)^*$
shows $(t \cdot \sigma, t \cdot \tau) \in (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

lemma *term-subst-rsteps-join*:

assumes $\bigwedge y. y \in \text{vars-term } u \implies (\sigma_1\ y, \sigma_2\ y) \in (\text{rstep } R)^\downarrow$
shows $(u \cdot \sigma_1, u \cdot \sigma_2) \in (\text{rstep } R)^\downarrow$
 $\langle \text{proof} \rangle$

lemma *funas-trs-converse* [simp]: $\text{funas-trs } (R^{-1}) = \text{funas-trs } R$

$\langle \text{proof} \rangle$

lemma *rstep-rev*: **assumes** $(s, t) \in \text{rstep-pos } \{(l, r)\}$ **p** **shows** $((t, s) \in \text{rstep-pos } \{(r, l)\})\ p$

$\langle \text{proof} \rangle$

lemma *conversion-ctxt-closed*: $(s, t) \in (\text{rstep } R)^{\leftrightarrow*} \implies (C\langle s \rangle, C\langle t \rangle) \in (\text{rstep } R)^{\leftrightarrow*}$

$R)^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *conversion-subst-closed*:
 $(s, t) \in (rstep\ R)^{\leftrightarrow*} \implies (s \cdot \sigma, t \cdot \sigma) \in (rstep\ R)^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *rstep-simulate-conv*:
assumes $\bigwedge l\ r. (l, r) \in S \implies (l, r) \in (rstep\ R)^{\leftrightarrow*}$
shows $(rstep\ S) \subseteq (rstep\ R)^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *symcl-simulate-conv*:
assumes $\bigwedge l\ r. (l, r) \in S \implies (l, r) \in (rstep\ R)^{\leftrightarrow*}$
shows $(rstep\ S)^{\leftrightarrow} \subseteq (rstep\ R)^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *conv-union-simulate*:
assumes $\bigwedge l\ r. (l, r) \in S \implies (l, r) \in (rstep\ R)^{\leftrightarrow*}$
shows $(rstep\ (R \cup S))^{\leftrightarrow*} = (rstep\ R)^{\leftrightarrow*}$
 $\langle proof \rangle$

definition *suptrel* $R = (relto\ \{\triangleright\}\ (rstep\ R))^+$

end

5 The Critical Pair Lemma

theory *Critical-Pairs*
imports
 Trs
 $First-Order-Terms.Unification$
begin

context
fixes $ren :: 'v :: infinite\ renaming2$
begin

definition
 $critical-Peaks :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs \Rightarrow (((('f, 'v)term \times ('f, 'v)term \times ('f, 'v)term))\ set$
where
 $critical-Peaks\ P\ R = \{ ((C \cdot_c \sigma) \langle r' \cdot \tau \rangle, l \cdot \sigma, r \cdot \sigma) \mid l\ r\ l'\ r'\ l''\ C\ \sigma\ \tau.$
 $(l, r) \in P \wedge (l', r') \in R \wedge l = C \langle l'' \rangle \wedge is-Fun\ l'' \wedge$
 $mgu-vd\ ren\ l''\ l' = Some\ (\sigma, \tau)\}$

definition

$critical-pairs :: ('f, 'v) trs \Rightarrow ('f, 'v) trs \Rightarrow (bool \times ('f, 'v) rule) set$

where

$critical-pairs\ P\ R = \{ (C = \square, (C \cdot_c \sigma)\langle r' \cdot \tau \rangle, r \cdot \sigma) \mid l\ r\ l'\ r'\ l''\ C\ \sigma\ \tau. \\ (l, r) \in P \wedge (l', r') \in R \wedge l = C\langle l'' \rangle \wedge is-Fun\ l'' \wedge \\ mgu-vd\ ren\ l''\ l' = Some\ (\sigma, \tau) \}$

lemma critical-pairsI:

assumes $(l, r) \in P$ **and** $(l', r') \in R$ **and** $l = C\langle l'' \rangle$
and $is-Fun\ l''$ **and** $mgu-vd\ ren\ l''\ l' = Some\ (\sigma, \tau)$ **and** $t = r \cdot \sigma$
and $s = (C \cdot_c \sigma)\langle r' \cdot \tau \rangle$ **and** $b = (C = \square)$
shows $(b, s, t) \in critical-pairs\ P\ R$
 $\langle proof \rangle$

lemma critical-pairs-mono:

assumes $S_1 \subseteq R_1$ **and** $S_2 \subseteq R_2$ **shows** $critical-pairs\ S_1\ S_2 \subseteq critical-pairs\ R_1\ R_2$
 $\langle proof \rangle$

lemma critical-Peaks-main:

fixes $P\ R :: ('f, 'v) trs$
assumes $tu: (t, u) \in rstep\ P$ **and** $ts: (t, s) \in rstep\ R$
shows $(s, u) \in (rstep\ R)^* \circ rrstep\ P \circ ((rstep\ R)^*)^* \circ -1 \vee$
 $(\exists\ C\ l\ m\ r\ \sigma. s = C\langle l \cdot \sigma \rangle \wedge t = C\langle m \cdot \sigma \rangle \wedge u = C\langle r \cdot \sigma \rangle \wedge$
 $(l, m, r) \in critical-Peaks\ P\ R)$
 $\langle proof \rangle$

lemma critical-Peaks-main-rrstep:

fixes $R :: ('f, 'v) trs$
assumes $tu: (t, u) \in rrstep\ R$ **and** $ts: (t, s) \in rstep\ R$
shows $(s, u) \in join\ (rstep\ R) \vee$
 $(\exists\ C\ l\ m\ r\ \sigma. s = C\langle l \cdot \sigma \rangle \wedge t = C\langle m \cdot \sigma \rangle \wedge u = C\langle r \cdot \sigma \rangle \wedge$
 $(l, m, r) \in critical-Peaks\ R\ R)$
 $\langle proof \rangle$

lemma parallel-rstep:

fixes $p1 :: pos$
assumes $p12: p1 \perp p2$
and $p1: p1 \in poss\ t$
and $p2: p2 \in poss\ t$
and $step2: t \vdash p2 = l2 \cdot \sigma2\ (l2, r2) \in R$
shows $(replace-at\ t\ p1\ v, replace-at\ (replace-at\ t\ p1\ v)\ p2\ (r2 \cdot \sigma2)) \in rstep\ R$
 $(is\ (?one, ?two) \in -)$
 $\langle proof \rangle$

lemma critical-Peaks-main-rstep:

fixes $R :: ('f, 'v) trs$
assumes $tu: (t, u) \in rstep\ R$ **and** $ts: (t, s) \in rstep\ R$
shows $(s, u) \in join\ (rstep\ R) \vee$

$(\exists C \ l \ m \ r \ \sigma. \ s = C\langle l \cdot \sigma \rangle \wedge t = C\langle m \cdot \sigma \rangle \wedge u = C\langle r \cdot \sigma \rangle \wedge$
 $((l, m, r) \in \text{critical-Peaks } R \ R \vee (r, m, l) \in \text{critical-Peaks } R \ R))$
 $\langle \text{proof} \rangle$

lemma critical-Peak-steps:
fixes $R :: ('f, 'v) \text{ trs}$ **and** S
assumes $cp: (l, m, r) \in \text{critical-Peaks } R \ S$
shows $(m, l) \in \text{rstep } S \ (m, r) \in \text{rstep } R \ (m, r) \in \text{rrstep } R$
 $\langle \text{proof} \rangle$

lemma critical-Peak-to-pair: **assumes** $(l, m, r) \in \text{critical-Peaks } R \ R$
shows $\exists b. (b, l, r) \in \text{critical-pairs } R \ R$
 $\langle \text{proof} \rangle$

lemma critical-pairs-main:
fixes $R :: ('f, 'v) \text{ trs}$
assumes $st1: (s, t1) \in \text{rstep } R$ **and** $st2: (s, t2) \in \text{rstep } R$
shows $(t1, t2) \in \text{join } (\text{rstep } R) \vee$
 $(\exists C \ b \ l \ r \ \sigma. \ t1 = C\langle l \cdot \sigma \rangle \wedge t2 = C\langle r \cdot \sigma \rangle \wedge$
 $((b, l, r) \in \text{critical-pairs } R \ R \vee (b, r, l) \in \text{critical-pairs } R \ R))$
 $\langle \text{proof} \rangle$

lemma critical-pairs:
fixes $R :: ('f, 'v) \text{ trs}$
assumes $cp: \bigwedge l \ r \ b. (b, l, r) \in \text{critical-pairs } R \ R \implies l \neq r \implies$
 $\exists l' \ r' \ s. \text{instance-rule } (l, r) \ (l', r') \wedge (l', s) \in (\text{rstep } R)^* \wedge (r', s) \in (\text{rstep } R)^*$
shows $WCR \ (\text{rstep } R)$
 $\langle \text{proof} \rangle$

lemma critical-pairs-fork:
fixes $R :: ('f, 'v) \text{ trs}$ **and** S
assumes $cp: (b, l, r) \in \text{critical-pairs } R \ S$
shows $(r, l) \in (\text{rstep } R)^{-1} \ O \ \text{rstep } S$
 $\langle \text{proof} \rangle$

lemma critical-pairs-fork': **assumes** $(b, l, r) \in \text{critical-pairs } R \ S$
shows $(l, r) \in (\text{rstep } S)^{\wedge -1} \ O \ \text{rstep } R$
 $\langle \text{proof} \rangle$

lemma critical-pairs-complete:
fixes $R :: ('f, 'v) \text{ trs}$
assumes $cp: (b, l, r) \in \text{critical-pairs } R \ R$
and $\text{no-join}: (l, r) \notin (\text{rstep } R)^{\downarrow}$
shows $\neg WCR \ (\text{rstep } R)$
 $\langle \text{proof} \rangle$

lemma critical-pair-lemma:

```

fixes  $R :: ('f, 'v) \text{trs}$ 
shows  $WCR \ (rstep \ R) \longleftrightarrow$ 
 $(\forall \ (b, s, t) \in \text{critical-pairs } R \ R. \ (s, t) \in (rstep \ R)^\downarrow)$ 
(is ?l = ?r)
 $\langle \text{proof} \rangle$ 

```

```

lemma critical-pairs-exI:
fixes  $\sigma :: ('f, 'v) \text{subst}$ 
assumes  $P: (l, r) \in P$  and  $R: (l', r') \in R$  and  $l: l = C\langle l' \rangle$ 
and  $l'': \text{is-Fun } l''$  and  $\text{unif}: l'' \cdot \sigma = l' \cdot \tau$ 
and  $b: b = (C = \square)$ 
shows  $\exists \ s \ t. (b, s, t) \in \text{critical-pairs } P \ R$ 
 $\langle \text{proof} \rangle$ 

```

```

end
end

```

```

theory SubList
imports
  HOL-Library.Sublist
  HOL-Library.Multiset
begin

```

```

lemmas subseq-trans = subseq-order.order-trans

```

```

lemma subseq-Cons-Cons:
assumes  $\text{subseq } (a \# as) \ (b \# bs)$ 
shows  $\text{subseq } as \ bs$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma subseq-induct2:
 $\llbracket \text{subseq } xs \ ys;$ 
 $\bigwedge \ bs. P \ \llbracket bs;$ 
 $\bigwedge \ a \ as \ bs. \llbracket \text{subseq } as \ bs; P \ as \ bs \rrbracket \implies P \ (a \# as) \ (a \# bs);$ 
 $\bigwedge \ a \ as \ b \ bs. \llbracket a \neq b; \text{subseq } as \ bs; \text{subseq } (a \# as) \ bs; P \ as \ bs; P \ (a \# as) \ bs \rrbracket$ 
 $\implies P \ (a \# as) \ (b \# bs) \rrbracket$ 
 $\implies P \ xs \ ys$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma subseq-submultiset:
 $\text{subseq } xs \ ys \implies \text{mset } xs \subseteq \# \text{mset } ys$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma subseq-subset:
 $\text{subseq } xs \ ys \implies \text{set } xs \subseteq \text{set } ys$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma remove1-subseq:
 $\text{subseq } (\text{remove1 } x \ xs) \ xs$ 

```

$\langle \text{proof} \rangle$

lemma *subseq-concat*:

assumes $\bigwedge x. x \in \text{set } xs \implies \text{subseq } (f \ x) \ (g \ x)$

shows $\text{subseq } (\text{concat } (\text{map } f \ xs)) \ (\text{concat } (\text{map } g \ xs))$

$\langle \text{proof} \rangle$

end

6 Multihole Contexts

theory *Multihole-Context*

imports

Trs

Preliminaries

SubList

begin

unbundle *lattice-syntax*

6.1 Partitioning lists into chunks of given length

fun *partition-by*

where

partition-by $xs \ [] = [] \mid$

partition-by $xs \ (y \# ys) = \text{take } y \ xs \ \# \ \text{partition-by } (\text{drop } y \ xs) \ ys$

lemma *partition-by-map0-append* [*simp*]:

$\text{partition-by } xs \ (\text{map } (\lambda x. 0) \ ys \ @ \ zs) = \text{replicate } (\text{length } ys) \ [] \ @ \ \text{partition-by } xs$

zs

$\langle \text{proof} \rangle$

lemma *concat-partition-by* [*simp*]:

$\text{sum-list } ys = \text{length } xs \implies \text{concat } (\text{partition-by } xs \ ys) = xs$

$\langle \text{proof} \rangle$

definition *partition-by-idx* **where**

partition-by-idx $l \ ys \ i \ j = \text{partition-by } [0..<l] \ ys \ ! \ i \ ! \ j$

lemma *partition-by-nth-nth-old*:

assumes $i < \text{length } (\text{partition-by } xs \ ys)$

and $j < \text{length } (\text{partition-by } xs \ ys \ ! \ i)$

and $\text{sum-list } ys = \text{length } xs$

shows $\text{partition-by } xs \ ys \ ! \ i \ ! \ j = xs \ ! \ (\text{sum-list } (\text{map } \text{length } (\text{take } i \ (\text{partition-by } xs \ ys)))) + j)$

$\langle \text{proof} \rangle$

lemma *map-map-partition-by*:

$\text{map } (\text{map } f) \ (\text{partition-by } xs \ ys) = \text{partition-by } (\text{map } f \ xs) \ ys$

$\langle \text{proof} \rangle$

lemma *length-partition-by* [simp]:
 $\text{length } (\text{partition-by } xs \ ys) = \text{length } ys$
 $\langle \text{proof} \rangle$

lemma *partition-by-Nil* [simp]:
 $\text{partition-by } [] \ ys = \text{replicate } (\text{length } ys) \ []$
 $\langle \text{proof} \rangle$

lemma *partition-by-concat-id* [simp]:
 assumes $\text{length } xss = \text{length } ys$
 and $\bigwedge i. i < \text{length } ys \implies \text{length } (xss ! i) = ys ! i$
 shows $\text{partition-by } (\text{concat } xss) \ ys = xss$
 $\langle \text{proof} \rangle$

lemma *partition-by-nth*:
 $i < \text{length } ys \implies \text{partition-by } xs \ ys ! i = \text{take } (ys ! i) \ (\text{drop } (\text{sum-list } (\text{take } i \ ys)))$
 xs
 $\langle \text{proof} \rangle$

lemma *partition-by-nth-less*:
 assumes $k < i$ **and** $i < \text{length } zs$
 and $\text{length } xs = \text{sum-list } (\text{take } i \ zs) + j$
 shows $\text{partition-by } (xs @ y \# ys) \ zs ! k = \text{take } (zs ! k) \ (\text{drop } (\text{sum-list } (\text{take } k \ zs))) \ xs$
 $\langle \text{proof} \rangle$

lemma *partition-by-nth-greater*:
 assumes $i < k$ **and** $k < \text{length } zs$ **and** $j < zs ! i$
 and $\text{length } xs = \text{sum-list } (\text{take } i \ zs) + j$
 shows $\text{partition-by } (xs @ y \# ys) \ zs ! k =$
 $\text{take } (zs ! k) \ (\text{drop } (\text{sum-list } (\text{take } k \ zs) - 1) \ (xs @ ys))$
 $\langle \text{proof} \rangle$

lemma *length-partition-by-nth*:
 $\text{sum-list } ys = \text{length } xs \implies i < \text{length } ys \implies \text{length } (\text{partition-by } xs \ ys ! i) = ys$
 $! i$
 $\langle \text{proof} \rangle$

lemma *partition-by-nth-nth-elem*:
 assumes $\text{sum-list } ys = \text{length } xs$ $i < \text{length } ys$ $j < ys ! i$
 shows $\text{partition-by } xs \ ys ! i ! j \in \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *partition-by-nth-nth*:
 assumes $\text{sum-list } ys = \text{length } xs$ $i < \text{length } ys$ $j < ys ! i$
 shows $\text{partition-by } xs \ ys ! i ! j = xs ! \text{partition-by-idx } (\text{length } xs) \ ys \ i \ j$

partition-by-idx (*length xs*) *ys i j < length xs*
 ⟨proof⟩

lemma *map-length-partition-by* [*simp*]:
sum-list ys = length xs \implies map length (partition-by xs ys) = ys
 ⟨proof⟩

lemma *map-partition-by-nth* [*simp*]:
i < length ys \implies map f (partition-by xs ys ! i) = partition-by (map f xs) ys ! i
 ⟨proof⟩

lemma *sum-list-partition-by* [*simp*]:
sum-list ys = length xs \implies
sum-list (map (λx . sum-list (map f x)) (partition-by xs ys)) = sum-list (map f xs)
 ⟨proof⟩

lemma *partition-by-map-conv*:
partition-by xs ys = map (λi . take (ys ! i) (drop (sum-list (take i ys)) xs)) [0 ..< length ys]
 ⟨proof⟩

lemma *UN-set-partition-by-map*:
sum-list ys = length xs \implies ($\bigcup_{x \in \text{set } (partition-by (map f xs) ys)}$ (set x)) = \bigcup (set (map f xs))
 ⟨proof⟩

lemma *UN-set-partition-by*:
sum-list ys = length xs \implies ($\bigcup_{zs \in \text{set } (partition-by xs ys)}$ $\bigcup_{x \in \text{set } zs}$ f x) = ($\bigcup_{x \in \text{set } xs}$ f x)
 ⟨proof⟩

lemma *Ball-atLeast0LessThan-partition-by-conv*:
($\forall i \in \{0..< \text{length } ys\}$. $\forall x \in \text{set } (partition-by xs ys ! i)$. $P x$) = ($\forall x \in \bigcup (\text{set } (\text{map set } (partition-by xs ys)))$. $P x$)
 ⟨proof⟩

lemma *Ball-set-partition-by*:
sum-list ys = length xs \implies
($\forall x \in \text{set } (partition-by xs ys)$. $\forall y \in \text{set } x$. $P y$) = ($\forall x \in \text{set } xs$. $P x$)
 ⟨proof⟩

lemma *partition-by-append2*:
partition-by xs (ys @ zs) = partition-by (take (sum-list ys) xs) ys @ partition-by (drop (sum-list ys) xs) zs
 ⟨proof⟩

lemma *partition-by-concat2*:
partition-by xs (concat ys) =

concat (map (λi . partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i))
 [0.. $\text{length } ys$])
 ⟨proof⟩

lemma partition-by-partition-by:

length xs = sum-list (map sum-list ys) \implies
 partition-by (partition-by xs (concat ys)) (map length ys) =
 map (λi. partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i)) [0.. length
 ys]
 ⟨proof⟩

datatype (f, vars-mctx : 'v) mctx = MVar 'v | MHole | MFun f (f, 'v) mctx
 list

6.2 Conversions from and to multihole contexts

primrec mctx-of-term :: (f, 'v) term \Rightarrow (f, 'v) mctx

where

mctx-of-term (Var x) = MVar x |
 mctx-of-term (Fun f ts) = MFun f (map mctx-of-term ts)

primrec term-of-mctx :: (f, 'v) mctx \Rightarrow (f, 'v) term

where

term-of-mctx (MVar x) = Var x |
 term-of-mctx (MFun f Cs) = Fun f (map term-of-mctx Cs)

lemma term-of-mctx-mctx-of-term-id [simp]:

term-of-mctx (mctx-of-term t) = t
 ⟨proof⟩

fun num-holes :: (f, 'v) mctx \Rightarrow nat

where

num-holes (MVar -) = 0 |
 num-holes MHole = 1 |
 num-holes (MFun - ctxts) = sum-list (map num-holes ctxts)

lemma num-holes-o-mctx-of-term [simp]:

num-holes \circ mctx-of-term = (λx. 0)
 ⟨proof⟩

lemma mctx-of-term-term-of-mctx-id [simp]:

num-holes C = 0 \implies mctx-of-term (term-of-mctx C) = C
 ⟨proof⟩

lemma vars-mctx-of-term [simp]: vars-mctx (mctx-of-term t) = vars-term t

⟨proof⟩

lemma num-holes-mctx-of-term [simp]:

$num\text{-}holes\ (mctx\text{-}of\text{-}term\ t) = 0$
 $\langle proof \rangle$

fun $funas\text{-}mctx :: ('f, 'v)\ mctx \Rightarrow 'f\ sig$

where

$funas\text{-}mctx\ (MFun\ f\ Cs) = \{(f, length\ Cs)\} \cup \bigcup (funas\text{-}mctx\ ' set\ Cs) \mid$
 $funas\text{-}mctx\ - = \{\}$

fun $funas\text{-}mctx\text{-}list :: ('f, 'v)\ mctx \Rightarrow ('f \times nat)\ list$

where

$funas\text{-}mctx\text{-}list\ (MFun\ f\ Cs) = (f, length\ Cs) \# concat\ (map\ funas\text{-}mctx\text{-}list\ Cs) \mid$
 $funas\text{-}mctx\text{-}list\ - = []$

lemma $funas\text{-}mctx\text{-}list\ [simp]:$

$set\ (funas\text{-}mctx\text{-}list\ C) = funas\text{-}mctx\ C$
 $\langle proof \rangle$

fun $split\text{-}term :: (('f, 'v)\ term \Rightarrow bool) \Rightarrow ('f, 'v)\ term \Rightarrow ('f, 'v)\ mctx \times ('f, 'v)\ term\ list$

where

$split\text{-}term\ P\ (Var\ x) = (if\ P\ (Var\ x)\ then\ (MHole, [Var\ x])\ else\ (MVar\ x, [])) \mid$
 $split\text{-}term\ P\ (Fun\ f\ ts) =$
 $(if\ P\ (Fun\ f\ ts)\ then\ (MHole, [Fun\ f\ ts])$
 $else\ let\ us = map\ (split\text{-}term\ P)\ ts\ in\ (MFun\ f\ (map\ fst\ us), concat\ (map\ snd\ us)))$

fun $cap\text{-}till :: (('f, 'v)\ term \Rightarrow bool) \Rightarrow ('f, 'v)\ term \Rightarrow ('f, 'v)\ mctx$

where

$cap\text{-}till\ P\ (Var\ x) = (if\ P\ (Var\ x)\ then\ MHole\ else\ MVar\ x) \mid$
 $cap\text{-}till\ P\ (Fun\ f\ ts) = (if\ P\ (Fun\ f\ ts)\ then\ MHole\ else\ MFun\ f\ (map\ (cap\text{-}till\ P)\ ts))$

fun $uncap\text{-}till :: (('f, 'v)\ term \Rightarrow bool) \Rightarrow ('f, 'v)\ term \Rightarrow ('f, 'v)\ term\ list$

where

$uncap\text{-}till\ P\ (Var\ x) = (if\ P\ (Var\ x)\ then\ [Var\ x]\ else\ []) \mid$
 $uncap\text{-}till\ P\ (Fun\ f\ ts) = (if\ P\ (Fun\ f\ ts)\ then\ [Fun\ f\ ts]\ else\ concat\ (map\ (uncap\text{-}till\ P)\ ts))$

lemma $split\text{-}term\ [simp]:$

$split\text{-}term\ P\ t = (cap\text{-}till\ P\ t, uncap\text{-}till\ P\ t)$
 $\langle proof \rangle$

definition $if\text{-}Fun\text{-}in\text{-}set\ F = (\lambda t. is\text{-}Var\ t \vee the\ (root\ t) \in F)$

lemma $if\text{-}Fun\text{-}in\text{-}set\text{-}sims\ [simp]:$

$if\text{-}Fun\text{-}in\text{-}set\ F\ (Var\ x)$
 $if\text{-}Fun\text{-}in\text{-}set\ F\ (Fun\ f\ ts) \longleftrightarrow (f, length\ ts) \in F$
 $is\text{-}Var\ t \implies if\text{-}Fun\text{-}in\text{-}set\ F\ t$

is-Fun $t \implies \text{if-Fun-in-set } F \ t \iff \text{the } (\text{root } t) \in F$
 $\langle \text{proof} \rangle$

lemma *if-Fun-in-set-mono*:

$F \subseteq G \implies \text{if-Fun-in-set } F \ t \implies \text{if-Fun-in-set } G \ t$
 $\langle \text{proof} \rangle$

abbreviation *split-term-funas* $F \equiv \text{split-term } (\text{if-Fun-in-set } F)$

abbreviation *cap-till-funas* $F \equiv \text{cap-till } (\text{if-Fun-in-set } F)$

abbreviation *uncap-till-funas* $F \equiv \text{uncap-till } (\text{if-Fun-in-set } F)$

lemma *if-Fun-in-set-uncap-till-funas*:

$A \subseteq B \implies \text{if-Fun-in-set } A \ t \implies \text{uncap-till-funas } B \ t = [t]$
 $\langle \text{proof} \rangle$

lemma *cap-till-funasD* $[dest]$:

$fn \in \text{funas-mctxt } (\text{cap-till-funas } F \ t) \implies fn \in F \implies \text{False}$
 $\langle \text{proof} \rangle$

lemma *cap-till-funas*:

$\forall fn \in \text{funas-mctxt } (\text{cap-till-funas } F \ t). \ fn \notin F$
 $\langle \text{proof} \rangle$

lemma *uncap-till*:

$\forall s \in \text{set } (\text{uncap-till } P \ t). \ P \ s$
 $\langle \text{proof} \rangle$

lemma *uncap-till-singleton*:

assumes $s \in \text{set } (\text{uncap-till } P \ t)$

shows $\text{uncap-till } P \ s = [s]$

$\langle \text{proof} \rangle$

lemma *uncap-till-idemp* $[simp]$:

$\text{map } (\text{uncap-till } P) (\text{uncap-till } P \ t) = \text{map } (\lambda s. [s]) (\text{uncap-till } P \ t)$
 $\langle \text{proof} \rangle$

lemma *uncap-till-Fun* $[simp]$:

$P \ (\text{Fun } f \ ts) \implies \text{uncap-till } P \ (\text{Fun } f \ ts) = [\text{Fun } f \ ts]$
 $\langle \text{proof} \rangle$

abbreviation *partition-holes* $xs \ Cs \equiv \text{partition-by } xs \ (\text{map num-holes } Cs)$

abbreviation *partition-holes-idx* $l \ Cs \equiv \text{partition-by-idx } l \ (\text{map num-holes } Cs)$

fun *fill-holes* $:: ('f, 'v) \text{ mctxt} \Rightarrow ('f, 'v) \text{ term list} \Rightarrow ('f, 'v) \text{ term}$

where

fill-holes $(MVar \ x) \ - = Var \ x \mid$

fill-holes $MHole \ [t] = t \mid$

fill-holes $(MFun \ f \ cs) \ ts = Fun \ f \ (\text{map } (\lambda i. \text{fill-holes } (cs \ ! \ i) \ ts))$

$(\text{partition-holes } ts \text{ } cs \text{ } ! \text{ } i)) [0 \text{ } ..< \text{length } cs])$

The following induction scheme provides the *MFun* case with the list argument split according to the argument contexts. This feature is quite delicate: its benefit can be destroyed by premature simplification using the *sum-list* $?ys = \text{length } ?xs \implies \text{concat } (\text{partition-by } ?xs \text{ } ?ys) = ?xs$ simplification rule.

lemma *fill-holes-induct2*[consumes 2, case-names *MHole MVar MFun*]:
fixes $P :: ('f, 'v) \text{mctxt} \Rightarrow 'a \text{list} \Rightarrow 'b \text{list} \Rightarrow \text{bool}$
assumes $\text{len1}: \text{num-holes } C = \text{length } xs$ **and** $\text{len2}: \text{num-holes } C = \text{length } ys$
and $\text{Hole}: \bigwedge x y. P \text{MHole } [x] [y]$
and $\text{Var}: \bigwedge v. P (\text{MVar } v) [] []$
and $\text{Fun}: \bigwedge f Cs xs ys. \text{sum-list } (\text{map num-holes } Cs) = \text{length } xs \implies$
 $\text{sum-list } (\text{map num-holes } Cs) = \text{length } ys \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-holes } xs \text{ } Cs ! i) (\text{partition-holes } ys$
 $Cs ! i)) \implies$
 $P (\text{MFun } f Cs) (\text{concat } (\text{partition-holes } xs \text{ } Cs)) (\text{concat } (\text{partition-holes } ys \text{ } Cs))$
shows $P C xs ys$
 $\langle \text{proof} \rangle$

lemma *fill-holes-induct*[consumes 1, case-names *MHole MVar MFun*]:
fixes $P :: ('f, 'v) \text{mctxt} \Rightarrow 'a \text{list} \Rightarrow \text{bool}$
assumes $\text{len}: \text{num-holes } C = \text{length } xs$
and $\text{Hole}: \bigwedge x. P \text{MHole } [x]$
and $\text{Var}: \bigwedge v. P (\text{MVar } v) [] []$
and $\text{Fun}: \bigwedge f Cs xs. \text{sum-list } (\text{map num-holes } Cs) = \text{length } xs \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-holes } xs \text{ } Cs ! i)) \implies$
 $P (\text{MFun } f Cs) (\text{concat } (\text{partition-holes } xs \text{ } Cs))$
shows $P C xs$
 $\langle \text{proof} \rangle$

lemma *funas-term-fill-holes-iff*: $\text{num-holes } C = \text{length } ts \implies$
 $g \in \text{funas-term } (\text{fill-holes } C \text{ } ts) \iff g \in \text{funas-mctxt } C \vee (\exists t \in \text{set } ts. g \in$
 $\text{funas-term } t)$
 $\langle \text{proof} \rangle$

lemma *fill-holes-MHole*:
 $\text{length } ts = 1 \implies ts ! 0 = u \implies \text{fill-holes } \text{MHole } ts = u$
 $\langle \text{proof} \rangle$

lemmas
 $\text{map-partition-holes-nth } [simp] =$
 $\text{map-partition-by-nth } [of - \text{map num-holes } Cs \text{ for } Cs, \text{unfolded length-map}] \text{ and}$
 $\text{length-partition-holes } [simp] =$
 $\text{length-partition-by } [of - \text{map num-holes } Cs \text{ for } Cs, \text{unfolded length-map}]$

lemma *length-partition-holes-nth* [simp]:

assumes $\text{sum-list } (\text{map num-holes } cs) = \text{length } ts$
and $i < \text{length } cs$
shows $\text{length } (\text{partition-holes } ts \text{ } cs ! i) = \text{num-holes } (cs ! i)$
 $\langle \text{proof} \rangle$

lemma *concat-partition-holes-upt*:
assumes $i \leq \text{length } cs$
shows $\text{concat } [\text{partition-holes } ts \text{ } cs ! j. j \leftarrow [0 ..< i]] =$
 $\text{take } (\text{sum-list } [\text{num-holes } (cs ! j). j \leftarrow [0 ..< i]]) \text{ } ts$
 $\langle \text{proof} \rangle$

lemma *partition-holes-step*:
 $\text{partition-holes } ts \text{ } (C \# Cs) = \text{take } (\text{num-holes } C) \text{ } ts \# \text{partition-holes } (\text{drop}$
 $(\text{num-holes } C) \text{ } ts) \text{ } Cs$
 $\langle \text{proof} \rangle$

lemma *partition-holes-map-ctxt*:
assumes $\text{length } cs = \text{length } ds$
and $\bigwedge i. i < \text{length } cs \implies \text{num-holes } (cs ! i) = \text{num-holes } (ds ! i)$
shows $\text{partition-holes } ts \text{ } cs = \text{partition-holes } ts \text{ } ds$
 $\langle \text{proof} \rangle$

lemma *partition-holes-concat-id*:
assumes $\text{length } sss = \text{length } cs$
and $\bigwedge i. i < \text{length } cs \implies \text{num-holes } (cs ! i) = \text{length } (sss ! i)$
shows $\text{partition-holes } (\text{concat } sss) \text{ } cs = sss$
 $\langle \text{proof} \rangle$

lemma *partition-holes-fill-holes-conv*:
 $\text{fill-holes } (\text{MFun } f \text{ } cs) \text{ } ts =$
 $\text{Fun } f \text{ } [\text{fill-holes } (cs ! i) \text{ } (\text{partition-holes } ts \text{ } cs ! i). i \leftarrow [0 ..< \text{length } cs]]$
 $\langle \text{proof} \rangle$

lemma *fill-holes-arbitrary*:
assumes $lCs: \text{length } Cs = \text{length } ts$
and $lss: \text{length } ss = \text{length } ts$
and $\text{rec}: \bigwedge i. i < \text{length } ts \implies \text{num-holes } (Cs ! i) = \text{length } (ss ! i) \wedge f \text{ } (Cs !$
 $i) \text{ } (ss ! i) = ts ! i$
shows $\text{map } (\lambda i. f \text{ } (Cs ! i) \text{ } (\text{partition-holes } (\text{concat } ss) \text{ } Cs ! i)) \text{ } [0 ..< \text{length } Cs]$
 $= ts$
 $\langle \text{proof} \rangle$

lemma *fill-holes-MFun*:
assumes $lCs: \text{length } Cs = \text{length } ts$
and $lss: \text{length } ss = \text{length } ts$
and $\text{rec}: \bigwedge i. i < \text{length } ts \implies \text{num-holes } (Cs ! i) = \text{length } (ss ! i) \wedge \text{fill-holes}$

$(Cs ! i) (ss ! i) = ts ! i$
shows *fill-holes* (MFun f Cs) (concat ss) = Fun f ts
 ⟨proof⟩

inductive

eq-fill ::
 $(f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ mtxt} \times (f, 'v) \text{ term list} \Rightarrow \text{bool } ((-/ =_f -) [51, 51] 50)$

where

eqfI [intro]: $t = \text{fill-holes } D \text{ ss} \implies \text{num-holes } D = \text{length } ss \implies t =_f (D, ss)$

lemma *fill-holes-inj*:

assumes $\text{num-holes } C = \text{length } ss$
and $\text{num-holes } C = \text{length } ts$
and $\text{fill-holes } C \text{ ss} = \text{fill-holes } C \text{ ts}$
shows $ss = ts$

⟨proof⟩

lemma *eqf-refl* [intro]:

$\text{num-holes } C = \text{length } ts \implies \text{fill-holes } C \text{ ts} =_f (C, ts)$

⟨proof⟩

lemma *eqfE*:

assumes $t =_f (D, ss)$ **shows** $t = \text{fill-holes } D \text{ ss}$ $\text{num-holes } D = \text{length } ss$

⟨proof⟩

lemma *eqf-MFunI*:

assumes $\text{length } sss = \text{length } Cs$
and $\text{length } ts = \text{length } Cs$
and $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$
shows $\text{Fun } f \text{ ts} =_f (\text{MFun } f \text{ Cs}, \text{concat } sss)$

⟨proof⟩

lemma *eqf-MFunE*:

assumes $s =_f (\text{MFun } f \text{ Cs}, ss)$
obtains $ts \text{ sss}$ **where** $s = \text{Fun } f \text{ ts}$ $\text{length } ts = \text{length } Cs$ $\text{length } sss = \text{length } Cs$
 $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$
 $ss = \text{concat } sss$

⟨proof⟩

lemma *eqf-MHoleE*:

assumes $s =_f (\text{MHole}, ss)$
shows $ss = [s]$

⟨proof⟩

fun *mctxt-of-ctxt* :: $(f, 'v) \text{ ctxt} \Rightarrow (f, 'v) \text{ mctxt}$

where

$\text{mctxt-of-ctxt } \text{Hole} = \text{MHole} \mid$
 $\text{mctxt-of-ctxt } (\text{More } f \text{ ss}_1 \text{ } C \text{ ss}_2) =$
 $\text{MFun } f (\text{map mctxt-of-term } ss_1 @ \text{mctxt-of-ctxt } C \# \text{map mctxt-of-term } ss_2)$

lemma *num-holes-mctxt-of-ctxt* [simp]:

num-holes (mctxt-of-ctxt *C*) = 1

⟨proof⟩

lemma *mctxt-of-term*: $t =_f$ (mctxt-of-term *t*, [])

⟨proof⟩

lemma *mctxt-of-ctxt* [simp]:

$C \langle t \rangle =_f$ (mctxt-of-ctxt *C*, [*t*])

⟨proof⟩

lemma *fill-holes-ctxt-main'*:

assumes *num-holes* *C* = Suc (length *bef* + length *aft*)

shows $\exists D. (\forall s. \text{fill-holes } C \text{ (bef @ s \# aft)} = D \langle s \rangle) \wedge (C = \text{MFun } f \text{ cs} \longrightarrow D \neq \square)$

⟨proof⟩

lemma *fill-holes-ctxt-main*:

assumes *num-holes* *C* = Suc (length *bef* + length *aft*)

shows $\exists D. \forall s. \text{fill-holes } C \text{ (bef @ s \# aft)} = D \langle s \rangle$

⟨proof⟩

lemma *fill-holes-ctxt*:

assumes *nh*: *num-holes* *C* = length *ss*

and *i*: *i* < length *ss*

obtains *D* **where** $\bigwedge s. \text{fill-holes } C \text{ (ss[i := s])} = D \langle s \rangle$

⟨proof⟩

fun *map-vars-mctxt* :: ($'v \Rightarrow 'w$) \Rightarrow ($'f, 'v$) mctxt \Rightarrow ($'f, 'w$) mctxt

where

map-vars-mctxt *vw* *MHole* = *MHole* |

map-vars-mctxt *vw* (*MVar* *v*) = (*MVar* (*vw* *v*)) |

map-vars-mctxt *vw* (*MFun* *f* *Cs*) = *MFun* *f* (*map* (*map-vars-mctxt* *vw*) *Cs*)

lemma *map-vars-mctxt-id* [simp]:

map-vars-mctxt ($\lambda x. x$) *C* = *C*

⟨proof⟩

lemma *num-holes-map-vars-mctxt* [simp]:

num-holes (*map-vars-mctxt* *vw* *C*) = *num-holes* *C*

⟨proof⟩

lemma *map-vars-term-eq-fill*:

$t =_f (C, ss) \implies \text{map-vars-term } vw \ t =_f (\text{map-vars-mctxt } vw \ C, \text{map} (\text{map-vars-term } vw) \ ss)$

⟨proof⟩

lemma *map-vars-term-fill-holes*:

assumes *nh*: *num-holes C = length ss*
shows *map-vars-term vw (fill-holes C ss) =*
fill-holes (map-vars-mctxt vw C) (map (map-vars-term vw) ss)
 $\langle \text{proof} \rangle$

lemma *split-term-egf*:
t =_f (cap-till P t, uncap-till P t)
 $\langle \text{proof} \rangle$

lemma *fill-holes-cap-till-uncap-till-id [simp]*:
fill-holes (cap-till P t) (uncap-till P t) = t
 $\langle \text{proof} \rangle$

lemma *num-holes-cap-till [simp]*:
num-holes (cap-till P t) = length (uncap-till P t)
 $\langle \text{proof} \rangle$

fun *split-vars* :: (*f*, *'v*) *term* \Rightarrow ((*f*, *'v*) *mctxt* \times *'v list*)
where
split-vars (Var x) = (MHole, [x]) |
split-vars (Fun f ts) = (MFun f (map (fst \circ split-vars) ts), concat (map (snd \circ split-vars) ts))

lemma *split-vars-num-holes*: *num-holes (fst (split-vars t)) = length (snd (split-vars t))*
 $\langle \text{proof} \rangle$

lemma *split-vars-vars-term-list*: *snd (split-vars t) = vars-term-list t*
 $\langle \text{proof} \rangle$

lemma *split-vars-vars-term*: *set (snd (split-vars t)) = vars-term t*
 $\langle \text{proof} \rangle$

lemma *split-vars-egf-subst-map-vars-term*:
t \cdot σ =_f (map-vars-mctxt vw (fst (split-vars t)), map σ (snd (split-vars t)))
 $\langle \text{proof} \rangle$

lemma *split-vars-egf-subst*: *t \cdot σ =_f (fst (split-vars t), (map σ (snd (split-vars t))))*
 $\langle \text{proof} \rangle$

lemma *split-vars-into-subst-map-vars-term*:
assumes *split*: *split-vars l = (C, xs)*
and *len*: *length ts = length xs*
and *id*: $\bigwedge i. i < \text{length } xs \implies \sigma (xs ! i) = ts ! i$
shows *l \cdot σ =_f (map-vars-mctxt vw C, ts)*
 $\langle \text{proof} \rangle$

lemma *split-vars-into-subst*:

assumes *split*: $\text{split-vars } l = (C, xs)$
and *len*: $\text{length } ts = \text{length } xs$
and *id*: $\bigwedge i. i < \text{length } xs \implies \sigma (xs ! i) = ts ! i$
shows $l \cdot \sigma =_f (C, ts)$
 $\langle \text{proof} \rangle$

lemma *eqf-funas-term*:
 $t =_f (C, ss) \implies \text{funas-term } t = \text{funas-mctxt } C \cup \bigcup (\text{funas-term } ' \text{ set } ss)$
 $\langle \text{proof} \rangle$

lemma *eqf-all-ctxt-closed-step*:
assumes *ctxt*: $\text{all-ctxt-closed } F \ R$
and *ass*: $t =_f (D, ss) \bigwedge i. i < \text{length } ts \implies (ss ! i, ts ! i) \in R \text{ length } ss = \text{length } ts \text{ funas-term } t \subseteq F$
 $\bigcup (\text{funas-term } ' \text{ set } ts) \subseteq F$
shows $(t, \text{fill-holes } D \ ts) \in R \wedge \text{fill-holes } D \ ts =_f (D, ts)$
 $\langle \text{proof} \rangle$

fun *map-mctxt* :: $(f \Rightarrow 'g) \Rightarrow ('f, 'v) \text{ mctxt} \Rightarrow ('g, 'v) \text{ mctxt}$
where
 $\text{map-mctxt } - (MVar \ x) = (MVar \ x) \mid$
 $\text{map-mctxt } - (MHole) = MHole \mid$
 $\text{map-mctxt } fg \ (MFun \ f \ Cs) = MFun \ (fg \ f) \ (\text{map } (\text{map-mctxt } fg) \ Cs)$

fun *ground-mctxt* :: $(f, 'v) \text{ mctxt} \Rightarrow \text{bool}$
where
 $\text{ground-mctxt } (MVar \ -) = \text{False} \mid$
 $\text{ground-mctxt } MHole = \text{True} \mid$
 $\text{ground-mctxt } (MFun \ f \ Cs) = \text{Ball } (\text{set } Cs) \ \text{ground-mctxt}$

lemma *ground-cap-till-funas* [intro]:
 $\text{ground-mctxt } (\text{cap-till-funas } F \ t)$
 $\langle \text{proof} \rangle$

lemma *ground-eq-fill*: $t =_f (C, ss) \implies \text{ground } t = (\text{ground-mctxt } C \wedge (\forall s \in \text{set } ss. \text{ground } s))$
 $\langle \text{proof} \rangle$

lemma *ground-fill-holes*:
assumes *nh*: $\text{num-holes } C = \text{length } ss$
shows $\text{ground } (\text{fill-holes } C \ ss) = (\text{ground-mctxt } C \wedge (\forall s \in \text{set } ss. \text{ground } s))$
 $\langle \text{proof} \rangle$

lemma *split-vars-ground*: $\text{split-vars } t = (C, xs) \implies \text{ground-mctxt } C$
 $\langle \text{proof} \rangle$

lemma *split-vars-ground-vars*:
assumes $\text{ground-mctxt } C$ **and** $\text{num-holes } C = \text{length } xs$
shows $\text{split-vars } (\text{fill-holes } C \ (\text{map } Var \ xs)) = (C, xs)$

$\langle \text{proof} \rangle$

lemma *ground-map-mtxt[simp]*: $\text{ground-mtxt } (\text{map-mtxt } fg \ C) = \text{ground-mtxt } C$
 $\langle \text{proof} \rangle$

lemma *num-holes-map-mtxt[simp]*: $\text{num-holes } (\text{map-mtxt } fg \ C) = \text{num-holes } C$
 $\langle \text{proof} \rangle$

lemma *split-vars-map-mtxt*:
 assumes *split*: $\text{split-vars } t = (\text{map-mtxt } fg \ C, \ xs)$
 shows $\text{split-vars } (\text{fill-holes } C \ (\text{map } \text{Var } xs)) = (C, \ xs)$
 $\langle \text{proof} \rangle$

lemma *subst-eq-map-decomp*:
 assumes $t \cdot \sigma = \text{map-funs-term } fg \ s$
 shows $\exists \ C \ xs \ \delta s. \ s =_f (C, \delta s) \wedge \text{split-vars } t = (\text{map-mtxt } fg \ C, \ xs) \wedge (\forall \ i < \text{length } xs. \sigma \ (xs \ ! \ i) = \text{map-funs-term } fg \ (\delta s \ ! \ i))$
 $\langle \text{proof} \rangle$

lemma *map-funs-term-fill-holes*:
 $\text{num-holes } C = \text{length } ss \implies$
 $\text{map-funs-term } fg \ (\text{fill-holes } C \ ss) =_f (\text{map-mtxt } fg \ C, \ \text{map } (\text{map-funs-term } fg) \ ss)$
 $\langle \text{proof} \rangle$

lemma *eqf-MVarE*:
 assumes $s =_f (MVar \ x, ss)$
 shows $s = Var \ x \ ss = []$
 $\langle \text{proof} \rangle$

lemma *eqf-imp-subt*:
 assumes $s: s =_f (C, ts)$
 and $t: t \in \text{set } ts$
 shows $s \supseteq t$
 $\langle \text{proof} \rangle$

lemma *eqf-MFun-imp-strict-subt*:
 assumes $s:s =_f (MFun \ f \ cs, \ ts)$
 and $t:t \in \text{set } ts$
 shows $s \supset t$
 $\langle \text{proof} \rangle$

fun *poss-mtxt* :: $(f, 'v) \text{ mtxt} \Rightarrow \text{pos set}$
where
 $\text{poss-mtxt } (MVar \ x) = \{[]\} \mid$
 $\text{poss-mtxt } MHole = \{\} \mid$

$\text{poss-mctxt} (\text{MFun } f \text{ cs}) = \{\emptyset\} \cup \bigcup (\text{set } (\text{map } (\lambda i. (\lambda p. i \# p) \text{ ' poss-mctxt } (cs \text{ ! } i)) [0 ..< \text{length } cs]))$

lemma *poss-mctxt-simp* [simp]:

$\text{poss-mctxt} (\text{MFun } f \text{ cs}) = \{\emptyset\} \cup \{i \# p \mid i p. i < \text{length } cs \wedge p \in \text{poss-mctxt } (cs \text{ ! } i)\}$

$\langle \text{proof} \rangle$

declare *poss-mctxt.simps*(3)[simp del]

lemma *poss-mctxt-map-vars-mctxt* [simp]:

$\text{poss-mctxt} (\text{map-vars-mctxt } f \text{ C}) = \text{poss-mctxt } C$

$\langle \text{proof} \rangle$

fun *hole-poss* :: ('f, 'v) mctxt \Rightarrow pos set

where

$\text{hole-poss } (\text{MVar } x) = \{\}$ |

$\text{hole-poss } \text{MHole} = \{\emptyset\}$ |

$\text{hole-poss } (\text{MFun } f \text{ cs}) = \bigcup (\text{set } (\text{map } (\lambda i. (\lambda p. i \# p) \text{ ' hole-poss } (cs \text{ ! } i)) [0 ..< \text{length } cs]))$

lemma *hole-poss-simp* [simp]:

$\text{hole-poss } (\text{MFun } f \text{ cs}) = \{i \# p \mid i p. i < \text{length } cs \wedge p \in \text{hole-poss } (cs \text{ ! } i)\}$

$\langle \text{proof} \rangle$

declare *hole-poss.simps*(3)[simp del]

lemma *hole-poss-empty-iff-num-holes-0*: $\text{hole-poss } C = \{\} \longleftrightarrow \text{num-holes } C = 0$

$\langle \text{proof} \rangle$

lemma *mctxt-of-term-fill-holes* [simp]:

$\text{fill-holes } (\text{mctxt-of-term } t) [] = t$

$\langle \text{proof} \rangle$

lemma *hole-pos-not-in-poss-mctxt*:

assumes $p \in \text{hole-poss } C$

shows $p \notin \text{poss-mctxt } C$

$\langle \text{proof} \rangle$

lemma *hole-pos-in-filled-fun-poss*:

assumes *is-Fun* t

shows $\text{hole-pos } E \in \text{fun-poss } ((E \cdot_c \sigma) \langle t \cdot \sigma \rangle)$

$\langle \text{proof} \rangle$

fun

subst-apply-mctxt :: ('f, 'v) mctxt \Rightarrow ('f, 'v, 'w) gsubst \Rightarrow ('f, 'w) mctxt (**infixl** \cdot_{mc} 67)

where

$\text{MHole} \cdot_{mc} - = \text{MHole}$ |

$(\text{MVar } x) \cdot_{mc} \sigma = \text{mctxt-of-term } (\sigma x)$ |

$(\text{MFun } f \text{ cs}) \cdot_{mc} \sigma = \text{MFun } f [c \cdot_{mc} \sigma \cdot c \leftarrow cs]$

lemma *subst-apply-mctxt-compose*: $C \cdot mc \sigma \cdot mc \delta = C \cdot mc \sigma \circ_s \delta$

<proof>

lemma *subst-apply-mctxt-cong*: $(\bigwedge x. x \in vars\text{-}mctxt\ C \implies \sigma x = \tau x) \implies C \cdot mc \sigma = C \cdot mc \tau$

<proof>

lemma *vars-mctxt-subst*: $vars\text{-}mctxt\ (C \cdot mc \sigma) = \bigcup (vars\text{-}term\ ' \sigma\ ' vars\text{-}mctxt\ C)$

<proof>

lemma *subst-apply-mctxt-numholes*:

shows $num\text{-}holes\ (c \cdot mc \sigma) = num\text{-}holes\ c$

<proof>

lemma *subst-apply-mctxt-fill-holes*:

assumes nh : $num\text{-}holes\ c = length\ ts$

shows $(fill\text{-}holes\ c\ ts) \cdot \sigma = fill\text{-}holes\ (c \cdot mc \sigma)\ [ti \cdot \sigma . ti \leftarrow ts]$

<proof>

lemma *subst-apply-mctxt-sound*:

assumes $t =_f\ (c, ts)$

shows $t \cdot \sigma =_f\ (c \cdot mc \sigma, [ti \cdot \sigma . ti \leftarrow ts])$

<proof>

fun *fill-holes-mctxt* :: $(f, 'v)\ mctxt \Rightarrow (f, 'v)\ mctxt\ list \Rightarrow (f, 'v)\ mctxt$

where

fill-holes-mctxt $(MVar\ x) = MVar\ x$ |

fill-holes-mctxt $MHole\ [] = MHole\ |$

fill-holes-mctxt $MHole\ [t] = t$ |

fill-holes-mctxt $(MFun\ f\ cs)\ ts = (MFun\ f\ (map\ (\lambda i. fill\text{-}holes\text{-}mctxt\ (cs\ !\ i)\ (partition\text{-}holes\ ts\ cs\ !\ i))\ [0 ..< length\ cs]))$

lemma *fill-holes-mctxt-Nil* [*simp*]:

fill-holes-mctxt $C\ [] = C$

<proof>

lemma *map-fill-holes-mctxt-zip* [*simp*]:

assumes $length\ ts = n$

shows $map\ (\lambda(x, y). fill\text{-}holes\text{-}mctxt\ x\ y)\ (zip\ (map\ mctxt\text{-}of\text{-}term\ ts)\ (replicate\ n\ [])) =$

$map\ mctxt\text{-}of\text{-}term\ ts$

<proof>

lemma *fill-holes-mctxt-MHole* [*simp*]:

$length\ ts = Suc\ 0 \implies fill\text{-}holes\text{-}mctxt\ MHole\ ts = hd\ ts$

<proof>

lemma *partition-holes-fill-holes-mctxt-conv*:

fill-holes-mctxt (MFun *f* *Cs*) *ts* =
 MFun *f* [*fill-holes-mctxt* (*Cs* ! *i*) (*partition-holes ts Cs* ! *i*). *i* ← [0 ..< length
Cs]]
 ⟨*proof*⟩

lemma *partition-holes-fill-holes-mctxt-conv'*:

fill-holes-mctxt (MFun *f* *Cs*) *ts* =
 MFun *f* (map (case-prod *fill-holes-mctxt*) (zip *Cs* (*partition-holes ts Cs*)))
 ⟨*proof*⟩

lemma *fill-holes-mctxt-mctxt-of-ctxt-mctxt-of-term [simp]*:

fill-holes-mctxt (*mctxt-of-ctxt C*) [*mctxt-of-term t*] = *mctxt-of-term* (*C*⟨*t*⟩)
 ⟨*proof*⟩

lemma *fill-holes-mctxt-mctxt-of-ctxt-MHole [simp]*:

fill-holes-mctxt (*mctxt-of-ctxt C*) [*MHole*] = *mctxt-of-ctxt C*
 ⟨*proof*⟩

lemma *partition-holes-fill-holes-conv'*:

fill-holes (MFun *f* *Cs*) *ts* =
 Fun *f* (map (case-prod *fill-holes*) (zip *Cs* (*partition-holes ts Cs*)))
 ⟨*proof*⟩

lemma *fill-holes-mctxt-MFun-replicate-length [simp]*:

fill-holes-mctxt (MFun *c* (*replicate* (length *Cs*) *MHole*)) *Cs* = MFun *c* *Cs*
 ⟨*proof*⟩

lemma *fill-holes-MFun-replicate-length [simp]*:

fill-holes (MFun *c* (*replicate* (length *ts*) *MHole*)) *ts* = Fun *c* *ts*
 ⟨*proof*⟩

lemma *funas-mctxt-fill-holes-mctxt [simp]*:

assumes *num-holes C = length Ds*
shows *funas-mctxt* (*fill-holes-mctxt C Ds*) = *funas-mctxt C* ∪ ⋃ (*set* (map *funas-mctxt Ds*))
 (**is** ?*f* *C Ds* = ?*g* *C Ds*)
 ⟨*proof*⟩

lemma *fill-holes-mctxt-MFun*:

assumes *lCs: length Cs = length ts*
and *lss: length ss = length ts*
and *rec: ⋀ i. i < length ts ⟹ num-holes (Cs ! i) = length (ss ! i) ∧*
fill-holes-mctxt (Cs ! i) (ss ! i) = ts ! i
shows *fill-holes-mctxt* (MFun *f* *Cs*) (*concat ss*) = MFun *f* *ts*
 ⟨*proof*⟩

lemma *num-holes-fill-holes-mctxt*:

assumes $\text{num-holes } C = \text{length } Ds$
shows $\text{num-holes } (\text{fill-holes-mctxt } C \ Ds) = \text{sum-list } (\text{map } \text{num-holes } Ds)$
 <proof>

lemma *fill-holes-mctxt-fill-holes*:
assumes $\text{len-ds}: \text{length } ds = \text{num-holes } c$
and $\text{nh}: \text{num-holes } (\text{fill-holes-mctxt } c \ ds) = \text{length } ss$
shows $\text{fill-holes } (\text{fill-holes-mctxt } c \ ds) \ ss =$
 $\text{fill-holes } c \ [\text{fill-holes } (ds \ ! \ i) \ (\text{partition-holes } ss \ ds \ ! \ i). \ i \leftarrow [0 \ ..< \ \text{num-holes } c]]$
 <proof>

lemma *fill-holes-mctxt-sound*:
assumes $\text{len-ds}: \text{length } ds = \text{num-holes } c$
and $\text{len-sss}: \text{length } sss = \text{num-holes } c$
and $\text{len-ts}: \text{length } ts = \text{num-holes } c$
and $\text{insts}: \bigwedge i. i < \text{length } ds \implies ts!i =_f (ds!i, sss!i)$
shows $\text{fill-holes } c \ ts =_f (\text{fill-holes-mctxt } c \ ds, \text{concat } sss)$
 <proof>

lemma *poss-mctxt-fill-holes-mctxt*:
assumes $p \in \text{poss-mctxt } C$
shows $p \in \text{poss-mctxt } (\text{fill-holes-mctxt } C \ Cs)$
 <proof>

fun *compose-mctxt* :: $(f, 'v) \text{ mctxt} \Rightarrow \text{nat} \Rightarrow (f, 'v) \text{ mctxt} \Rightarrow (f, 'v) \text{ mctxt}$
where
 $\text{compose-mctxt } C \ i \ Ci =$
 $\text{fill-holes-mctxt } C \ [(if \ i = j \ \text{then } Ci \ \text{else } MHole). \ j \leftarrow [0 \ ..< \ \text{num-holes } C]]$

lemma *funas-mctxt-compose-mctxt* [*simp*]:
assumes $i < \text{num-holes } C$
shows $\text{funas-mctxt } (\text{compose-mctxt } C \ i \ D) = \text{funas-mctxt } C \cup \text{funas-mctxt } D$
 <proof>

lemma *compose-mctxt-sound*:
assumes $s: s =_f (C, \text{bef } @ \ si \ \# \ \text{aft})$
and $si: si =_f (Ci, ts)$
and $i: i = \text{length } \text{bef}$
shows $s =_f (\text{compose-mctxt } C \ i \ Ci, \text{bef } @ \ ts \ @ \ \text{aft})$
 <proof>

fun *mctxt-fill-partially-mctxts* :: $(f, 'v) \text{ term list} \Rightarrow (f, 'v) \text{ term list} \Rightarrow (f, 'v) \text{ mctxt list}$

where
 $\text{mctxt-fill-partially-mctxts } [] \ ts = \text{map } \text{mctxt-of-term } ts \mid$
 $\text{mctxt-fill-partially-mctxts } (s \ \# \ ss) \ (t \ \# \ ts) =$
 $(if \ s = t \ \text{then } (MHole \ \# \ \text{mctxt-fill-partially-mctxts } ss \ ts)$
 $\text{else } (\text{mctxt-of-term } t \ \# \ \text{mctxt-fill-partially-mctxts } (s \ \# \ ss) \ ts))$

fun

mctxt-fill-partially-fills ::
 (*f*, *'v*) *term list* \Rightarrow (*f*, *'v*) *term list* \Rightarrow (*f*, *'v*) *term list list*

where

mctxt-fill-partially-fills [] *ts* = *map* (*const* []) *ts* |
mctxt-fill-partially-fills (*s* # *ss*) (*t* # *ts*) =
 (if *s* = *t* then ([*s*] # *mctxt-fill-partially-fills* *ss* *ts*)
 else ([] # *mctxt-fill-partially-fills* (*s* # *ss*) *ts*))

lemma *mctxt-fill-partially-mctxts-length* [*simp*]:

assumes *subseq ss ts*
shows *length* (*mctxt-fill-partially-mctxts* *ss* *ts*) = *length ts*
 <*proof*>

lemma *mctxt-fill-partially-fills-length* [*simp*]:

assumes *subseq ss ts*
shows *length* (*mctxt-fill-partially-fills* *ss* *ts*) = *length ts*
 <*proof*>

lemma *mctxt-fill-partially-numholes*:

assumes *subseq ss ts*
shows *sum-list* [*num-holes ci . ci* \leftarrow *mctxt-fill-partially-mctxts* *ss* *ts*] = *length ss*
 <*proof*>

lemma *mctxt-fill-partially-sound*:

assumes *sl: subseq ss ts*
shows $\bigwedge i. i < \text{length } ts \implies ts!i =_f (\text{mctxt-fill-partially-mctxts } ss \text{ } ts ! i, \text{mctxt-fill-partially-fills } ss \text{ } ts ! i)$
 <*proof*>

lemma *mctxt-fill-partially*:

assumes *ss: subseq ss ts*
and *t: t* =_{*f*} (*c*, *ts*)
shows $\exists d. t =_f (d, ss)$
 <*proof*>

lemma *fill-holes-mctxt-map-mctxt-of-term-conv* [*simp*]:

assumes *num-holes C* = *length ts*
shows *fill-holes-mctxt C* (*map mctxt-of-term ts*) = *mctxt-of-term* (*fill-holes C ts*)
 <*proof*>

lemma *fill-holes-mctxt-of-ctxt* [*simp*]:

fill-holes (*mctxt-of-ctxt C*) [*t*] = *C* <*t*>
 <*proof*>

definition

compose-cap-till P t i C =
fill-holes-mctxt (*cap-till P t*) (*map mctxt-of-term* (*take i* (*uncap-till P t*)) @
C # *map mctxt-of-term* (*drop* (*Suc i*) (*uncap-till P t*)))

abbreviation $\text{compose-cap-till-funas } F \equiv \text{compose-cap-till } (\text{if-Fun-in-set } F)$

lemma *fill-holes-compose-cap-till*:

assumes $i < \text{num-holes } (\text{cap-till } P \ s)$ **and** $\text{num-holes } C = \text{length } ts$
shows $\text{fill-holes } (\text{compose-cap-till } P \ s \ i \ C) \ ts =$
 $\text{fill-holes } (\text{cap-till } P \ s) \ (\text{take } i \ (\text{uncap-till } P \ s) \ @ \ \text{fill-holes } C \ ts \ \# \ \text{drop } (\text{Suc } i) \ (\text{uncap-till } P \ s))$
(is $- = \text{fill-holes } - \ ?ss)$
 $\langle \text{proof} \rangle$

lemma *in-uncap-till-funas*:

assumes $\text{root: root } u = \text{Some } fn \ fn \in F$
and $t = C \langle u \rangle$
shows $\exists i < \text{length } (\text{uncap-till-funas } F \ t). \exists D. \text{uncap-till-funas } F \ t \ ! \ i = D \langle u \rangle \wedge$
 $\text{mctxt-of-ctxt } C = \text{compose-cap-till-funas } F \ t \ i \ (\text{mctxt-of-ctxt } D)$
 $\langle \text{proof} \rangle$

lemma *uncap-till-funas-fill-holes-cancel [simp]*:

assumes $\text{num-holes } C = \text{length } ts$ **and** $\text{ground-mctxt } C$
and $\text{funas-mctxt } C \subseteq - \ F$
shows $\text{uncap-till-funas } F \ (\text{fill-holes } C \ ts) = \text{concat } (\text{map } (\text{uncap-till-funas } F) \ ts)$
 $\langle \text{proof} \rangle$

lemma *uncap-till-funas-fill-holes-cap-till-funas [simp]*:

assumes $\text{num-holes } (\text{cap-till-funas } F \ s) = \text{length } ts$
shows $\text{uncap-till-funas } F \ (\text{fill-holes } (\text{cap-till-funas } F \ s) \ ts) =$
 $\text{concat } (\text{map } (\text{uncap-till-funas } F) \ ts)$
 $\langle \text{proof} \rangle$

lemma *Ball-atLeast0LessThan-partition-holes-conv [simp]*:

$(\forall i \in \{0 \ ..< \text{length } Cs\}. \forall x \in \text{set } (\text{partition-holes } xs \ Cs \ ! \ i). P \ x) =$
 $(\forall x \in \bigcup (\text{set } (\text{map } \text{set } (\text{partition-holes } xs \ Cs)))) . P \ x)$
 $\langle \text{proof} \rangle$

lemma *ground-fill-holes-mctxt [simp]*:

$\text{num-holes } C = \text{length } Ds \implies$
 $\text{ground-mctxt } (\text{fill-holes-mctxt } C \ Ds) \longleftrightarrow \text{ground-mctxt } C \wedge (\forall D \in \text{set } Ds. \text{ground-mctxt } D)$
 $\langle \text{proof} \rangle$

lemma *concat-map-uncap-till-funas-map-subst-apply-uncap-till-funas [simp]*:

$\text{concat } (\text{map } (\text{uncap-till-funas } F) \ (\text{map } (\lambda s. s \cdot \sigma) \ (\text{uncap-till-funas } F \ t))) =$
 $\text{uncap-till-funas } F \ (t \cdot \sigma)$
 $\langle \text{proof} \rangle$

lemma *concat-uncap-till-subst-conv*:

$\text{concat } (\text{map } (\lambda i. \text{uncap-till-funas } F \ ((\text{uncap-till-funas } F \ t \ ! \ i) \cdot \sigma)) \ [0 \ ..< \text{length } (\text{uncap-till-funas } F \ t)]) =$

uncap-till-funas $F (t \cdot \sigma)$
 $\langle \text{proof} \rangle$

lemma *the-root-uncap-till-funas*:
 $\text{is-Fun } t \implies \text{the } (\text{root } t) \in F \implies \text{uncap-till-funas } F t = [t]$
 $\langle \text{proof} \rangle$

lemma *funas-cap-till-subset*:
 $\text{funas-mctxt } (\text{cap-till } P t) \subseteq \text{funas-term } t$
 $\langle \text{proof} \rangle$

lemma *funas-uncap-till-subset*:
 $s \in \text{set } (\text{uncap-till } P t) \implies \text{funas-term } s \subseteq \text{funas-term } t$
 $\langle \text{proof} \rangle$

lemma *ground-mctxt-subst-apply-context* [simp]:
 $\text{ground-mctxt } C \implies C \cdot \text{mc } \sigma = C$
 $\langle \text{proof} \rangle$

lemma *vars-term-fill-holes* [simp]:
 $\text{num-holes } C = \text{length } ts \implies \text{ground-mctxt } C \implies$
 $\text{vars-term } (\text{fill-holes } C ts) = \bigcup (\text{vars-term } ' \text{ set } ts)$
 $\langle \text{proof} \rangle$

6.3 Semilattice Structures

instantiation *mctxt* :: (type, type) inf
begin

fun *inf-mctxt* :: ('a, 'b) mctxt \Rightarrow ('a, 'b) mctxt \Rightarrow ('a, 'b) mctxt
where

$M\text{Hole} \sqcap D = M\text{Hole} \mid$
 $C \sqcap M\text{Hole} = M\text{Hole} \mid$
 $M\text{Var } x \sqcap M\text{Var } y = (\text{if } x = y \text{ then } M\text{Var } x \text{ else } M\text{Hole}) \mid$
 $M\text{Fun } f \text{ } Cs \sqcap M\text{Fun } g \text{ } Ds =$
 $(\text{if } f = g \wedge \text{length } Cs = \text{length } Ds \text{ then } M\text{Fun } f \text{ } (\text{map } (\text{case-prod } (\sqcap)) (\text{zip } Cs$
 $Ds))$
 $\text{else } M\text{Hole}) \mid$
 $C \sqcap D = M\text{Hole}$

instance $\langle \text{proof} \rangle$

end

lemma *inf-mctxt-idem* [simp]:
fixes $C :: ('f, 'v) \text{ mctxt}$
shows $C \sqcap C = C$
 $\langle \text{proof} \rangle$

lemma *inf-mctxt-MHole2* [*simp*]:

$C \sqcap MHole = MHole$

$\langle proof \rangle$

lemma *inf-mctxt-comm* [*ac-simps*]:

$(C :: ('f, 'v) mctxt) \sqcap D = D \sqcap C$

$\langle proof \rangle$

lemma *inf-mctxt-assoc* [*ac-simps*]:

fixes $C :: ('f, 'v) mctxt$

shows $C \sqcap D \sqcap E = C \sqcap (D \sqcap E)$

$\langle proof \rangle$

instantiation *mctxt* :: (*type*, *type*) *order*

begin

definition $(C :: ('a, 'b) mctxt) \leq D \longleftrightarrow C \sqcap D = C$

definition $(C :: ('a, 'b) mctxt) < D \longleftrightarrow C \leq D \wedge \neg D \leq C$

instance

$\langle proof \rangle$

end

inductive *less-eq-mctxt'* :: (*f*, *v*) *mctxt* \Rightarrow (*f*, *v*) *mctxt* \Rightarrow *bool* **where**

less-eq-mctxt' *MHole* *u*

| *less-eq-mctxt'* (*MVar* *v*) (*MVar* *v*)

| *length* *cs* = *length* *ds* $\Longrightarrow (\bigwedge i. i < \text{length } cs \Longrightarrow \text{less-eq-mctxt}' (cs ! i) (ds ! i))$

$\Longrightarrow \text{less-eq-mctxt}' (MFun f cs) (MFun f ds)$

lemma *less-eq-mctxt-prime*: $C \leq D \longleftrightarrow \text{less-eq-mctxt}' C D$

$\langle proof \rangle$

lemmas *less-eq-mctxt-induct* = *less-eq-mctxt'.induct*[folded *less-eq-mctxt-prime*, *consumes* 1]

lemmas *less-eq-mctxt-intros* = *less-eq-mctxt'.intros*[folded *less-eq-mctxt-prime*]

lemma *less-eq-mctxtI2*:

$C = MHole \Longrightarrow C \leq MHole$

$C = MHole \vee C = MVar v \Longrightarrow C \leq MVar v$

$C = MHole \vee C = MFun f cs \wedge \text{length } cs = \text{length } ds \wedge (\forall i. i < \text{length } cs \longrightarrow cs ! i \leq ds ! i) \Longrightarrow C \leq MFun f ds$

$\langle proof \rangle$

lemma *less-eq-mctxt-MHoleE2*:

assumes $C \leq MHole$

obtains (*MHole*) $C = MHole$

$\langle proof \rangle$

lemma *less-eq-mctxt-MVarE2*:

assumes $C \leq MVar\ v$

obtains $(MHole)\ C = MHole \mid (MVar)\ C = MVar\ v$

$\langle proof \rangle$

lemma *less-eq-mctxt-MFunE2*:

assumes $C \leq MFun\ f\ ds$

obtains $(MHole)\ C = MHole$

$\mid (MFun)\ cs\ \mathbf{where}\ C = MFun\ f\ cs\ length\ cs = length\ ds \wedge i. i < length\ cs \implies$
 $cs\ !\ i \leq ds\ !\ i$

$\langle proof \rangle$

lemmas *less-eq-mctxtE2* = *less-eq-mctxt-MHoleE2* *less-eq-mctxt-MVarE2* *less-eq-mctxt-MFunE2*

lemma *less-eq-mctxtI1*:

$MHole \leq D$

$D = MVar\ v \implies MVar\ v \leq D$

$D = MFun\ f\ ds \implies length\ cs = length\ ds \implies (\bigwedge i. i < length\ cs \implies cs\ !\ i \leq ds$
 $\ !\ i) \implies MFun\ f\ cs \leq D$

$\langle proof \rangle$

lemma *less-eq-mctxt-MVarE1*:

assumes $MVar\ v \leq D$

obtains $(MVar)\ D = MVar\ v$

$\langle proof \rangle$

lemma *less-eq-mctxt-MFunE1*:

assumes $MFun\ f\ cs \leq D$

obtains $(MFun)\ ds\ \mathbf{where}\ D = MFun\ f\ ds\ length\ cs = length\ ds \wedge i. i < length$
 $cs \implies cs\ !\ i \leq ds\ !\ i$

$\langle proof \rangle$

lemmas *less-eq-mctxtE1* = *less-eq-mctxt-MVarE1* *less-eq-mctxt-MFunE1*

instance *mctxt* :: $(type, type)\ semilattice\text{-}inf$

$\langle proof \rangle$

fun *inf-mctxt-args* :: $(f, 'v)\ mctxt \Rightarrow (f, 'v)\ mctxt \Rightarrow (f, 'v)\ mctxt\ list$

where

inf-mctxt-args $MHole\ D = [MHole] \mid$

inf-mctxt-args $C\ MHole = [C] \mid$

inf-mctxt-args $(MVar\ x)\ (MVar\ y) = (if\ x = y\ then\ []\ else\ [MVar\ x]) \mid$

inf-mctxt-args $(MFun\ f\ Cs)\ (MFun\ g\ Ds) =$

$(if\ f = g \wedge length\ Cs = length\ Ds\ then\ concat\ (map\ (case\text{-}prod\ inf\text{-}mctxt\text{-}args)$
 $(zip\ Cs\ Ds))$

$else\ [MFun\ f\ Cs]) \mid$

inf-mctxt-args $C\ D = [C]$

lemma *inf-mctxt-args-MHole2* [simp]:
 $\text{inf-mctxt-args } C \text{ MHole} = [C]$
 ⟨proof⟩

lemma *fill-holes-mctxt-replicate-MHole* [simp]:
 $\text{fill-holes-mctxt } C (\text{replicate } (\text{num-holes } C) \text{ MHole}) = C$
 ⟨proof⟩

lemma *num-holes-inf-mctxt*:
 $\text{num-holes } (C \sqcap D) = \text{length } (\text{inf-mctxt-args } C \ D)$
 ⟨proof⟩

lemma *length-inf-mctxt-args*:
 $\text{length } (\text{inf-mctxt-args } D \ C) = \text{length } (\text{inf-mctxt-args } C \ D)$
 ⟨proof⟩

lemma *inf-mctxt-args-same* [simp]:
 $\text{inf-mctxt-args } C \ C = \text{replicate } (\text{num-holes } C) \text{ MHole}$
 ⟨proof⟩

lemma *inf-mctxt-inf-mctxt-args*:
 $\text{fill-holes-mctxt } (C \sqcap D) (\text{inf-mctxt-args } C \ D) = C$
 ⟨proof⟩

lemma *inf-mctxt-inf-mctxt-args2*:
 $\text{fill-holes-mctxt } (C \sqcap D) (\text{inf-mctxt-args } D \ C) = D$
 ⟨proof⟩

instantiation *mctxt* :: (type, type) sup
begin

fun *sup-mctxt* :: ('a, 'b) mctxt \Rightarrow ('a, 'b) mctxt \Rightarrow ('a, 'b) mctxt
where

$\text{MHole} \sqcup D = D \mid$
 $C \sqcup \text{MHole} = C \mid$
 $\text{MVar } x \sqcup \text{MVar } y = (\text{if } x = y \text{ then } \text{MVar } x \text{ else undefined}) \mid$
 $\text{MFun } f \ Cs \sqcup \text{MFun } g \ Ds =$
 $(\text{if } f = g \wedge \text{length } Cs = \text{length } Ds \text{ then } \text{MFun } f (\text{map } (\text{case-prod } (\sqcup)) (\text{zip } Cs$
 $Ds))$
 $\text{else undefined}) \mid$
 $(C :: ('a, 'b) \text{ mctxt}) \sqcup D = \text{undefined}$

instance ⟨proof⟩

end

lemma *sup-mctxt-idem* [simp]:
fixes $C :: ('f, 'v) \text{ mctxt}$
shows $C \sqcup C = C$

$\langle \text{proof} \rangle$

lemma *sup-mctxt-MHole* [*simp*]: $C \sqcup \text{MHole} = C$
 $\langle \text{proof} \rangle$

lemma *sup-mctxt-comm* [*ac-simps*]:
fixes $C :: ('f, 'v) \text{mctxt}$
shows $C \sqcup D = D \sqcup C$
 $\langle \text{proof} \rangle$

(\sqcup) is defined on compatible multihole-contexts. Note that compatibility is not transitive.

inductive-set *comp-mctxt* :: $(('a, 'b) \text{mctxt} \times ('a, 'b) \text{mctxt}) \text{ set}$
where

$\text{MHole1}: (\text{MHole}, D) \in \text{comp-mctxt} \mid$
 $\text{MHole2}: (C, \text{MHole}) \in \text{comp-mctxt} \mid$
 $\text{MVar}: x = y \implies (\text{MVar } x, \text{MVar } y) \in \text{comp-mctxt} \mid$
 $\text{MFun}: f = g \implies \text{length } Cs = \text{length } Ds \implies \forall i < \text{length } Ds. (Cs ! i, Ds ! i) \in \text{comp-mctxt} \implies$
 $(\text{MFun } f \text{ } Cs, \text{MFun } g \text{ } Ds) \in \text{comp-mctxt}$

lemma *comp-mctxt-refl*:
 $(C, C) \in \text{comp-mctxt}$
 $\langle \text{proof} \rangle$

lemma *comp-mctxt-sym*:
assumes $(C, D) \in \text{comp-mctxt}$
shows $(D, C) \in \text{comp-mctxt}$
 $\langle \text{proof} \rangle$

lemma *sup-mctxt-assoc* [*ac-simps*]:
assumes $(C, D) \in \text{comp-mctxt}$ **and** $(D, E) \in \text{comp-mctxt}$
shows $C \sqcup D \sqcup E = C \sqcup (D \sqcup E)$
 $\langle \text{proof} \rangle$

No instantiation to *semilattice-sup* possible, since (\sqcup) is only partially defined on terms (e.g., it is not associative in general).

interpretation *mctxt-order-bot*: *order-bot* *MHole* (\leq) $(<)$
 $\langle \text{proof} \rangle$

lemma *sup-mctxt-ge1* [*simp*]:
assumes $(C, D) \in \text{comp-mctxt}$
shows $C \leq C \sqcup D$
 $\langle \text{proof} \rangle$

lemma *sup-mctxt-ge2* [*simp*]:
assumes $(C, D) \in \text{comp-mctxt}$
shows $D \leq C \sqcup D$
 $\langle \text{proof} \rangle$

lemma *sup-mtxtt-least*:
 assumes $(D, E) \in \text{comp-mtxtt}$
 and $D \leq C$ and $E \leq C$
 shows $D \sqcup E \leq C$
 $\langle \text{proof} \rangle$

lemma *inf-mtxtt-args-MHole*:
 assumes $(C, D) \in \text{comp-mtxtt}$ and $i < \text{length} (\text{inf-mtxtt-args } C \ D)$
 shows $\text{inf-mtxtt-args } C \ D \ ! \ i = \text{MHole} \vee \text{inf-mtxtt-args } D \ C \ ! \ i = \text{MHole}$
 $\langle \text{proof} \rangle$

Parallel rewriting is closed under multihole-contexts.

lemma *par-rstep-mtxtt*:
 assumes $s =_f (C, ss)$ and $t =_f (C, ts)$
 and $\forall i < \text{length } ss. (ss \ ! \ i, ts \ ! \ i) \in \text{par-rstep } R$
 shows $(s, t) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mtxttD*:
 assumes $(s, t) \in \text{par-rstep } R$
 shows $\exists C \ ss \ ts. s =_f (C, ss) \wedge t =_f (C, ts) \wedge (\forall i < \text{length } ss. (ss \ ! \ i, ts \ ! \ i) \in \text{rrstep } R)$
 (is $\exists C \ ss \ ts. ?P \ s \ t \ C \ ss \ ts$)
 $\langle \text{proof} \rangle$

lemma *rsteps-mtxtt*:
 assumes $s =_f (C, ss)$ and $t =_f (C, ts)$
 and $\forall i < \text{length } ss. (ss \ ! \ i, ts \ ! \ i) \in (\text{rstep } R)^*$
 shows $(s, t) \in (\text{rstep } R)^*$
 $\langle \text{proof} \rangle$

fun *sup-mtxtt-args* :: $(f, 'v) \text{ mtxtt} \Rightarrow (f, 'v) \text{ mtxtt} \Rightarrow (f, 'v) \text{ mtxtt list}$
where

sup-mtxtt-args *MHole* $D = [D] \mid$
sup-mtxtt-args $C \ \text{MHole} = \text{replicate } (\text{num-holes } C) \ \text{MHole} \mid$
sup-mtxtt-args $(MVar \ x) \ (MVar \ y) = (\text{if } x = y \text{ then } [] \text{ else undefined}) \mid$
sup-mtxtt-args $(MFun \ f \ Cs) \ (MFun \ g \ Ds) =$
 (if $f = g \wedge \text{length } Cs = \text{length } Ds$ then $\text{concat } (\text{map } (\text{case-prod } \text{sup-mtxtt-args})$
 $(\text{zip } Cs \ Ds))$
 else undefined) \mid
sup-mtxtt-args $C \ D = \text{undefined}$

lemma *sup-mtxtt-args-MHole2 [simp]*:
sup-mtxtt-args $C \ \text{MHole} = \text{replicate } (\text{num-holes } C) \ \text{MHole}$
 $\langle \text{proof} \rangle$

lemma *num-holes-sup-mtxt-args*:

assumes $(C, D) \in \text{comp-mtxt}$

shows $\text{num-holes } C = \text{length } (\text{sup-mtxt-args } C \ D)$

$\langle \text{proof} \rangle$

lemma *sup-mtxt-sup-mtxt-args*:

assumes $(C, D) \in \text{comp-mtxt}$

shows $\text{fill-holes-mtxt } C \ (\text{sup-mtxt-args } C \ D) = C \sqcup D$

$\langle \text{proof} \rangle$

lemma *sup-mtxt-args*:

assumes $(C, D) \in \text{comp-mtxt}$

shows $\text{sup-mtxt-args } C \ D = \text{inf-mtxt-args } (C \sqcup D) \ C$

$\langle \text{proof} \rangle$

lemma *term-for-mtxt*:

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

obtains t **and** ts **where** $t =_f (C, ts)$

$\langle \text{proof} \rangle$

lemma *comp-mtxt-eqfE*:

assumes $(C, D) \in \text{comp-mtxt}$

obtains s **and** ss **and** ts **where** $s =_f (C, ss)$ **and** $s =_f (D, ts)$

$\langle \text{proof} \rangle$

lemma *eqf-comp-mtxt*:

assumes $s =_f (C, ss)$ **and** $s =_f (D, ts)$

shows $(C, D) \in \text{comp-mtxt}$

$\langle \text{proof} \rangle$

lemma *comp-mtxt-iff*:

$(C, D) \in \text{comp-mtxt} \longleftrightarrow (\exists s \ ss \ ts. s =_f (C, ss) \wedge s =_f (D, ts))$

$\langle \text{proof} \rangle$

lemma *hole-poss-parallel-pos [simp]*:

assumes $p \in \text{hole-poss } C$ **and** $q \in \text{hole-poss } C$ **and** $p \neq q$

shows $\text{parallel-pos } p \ q$

$\langle \text{proof} \rangle$

lemma *eq-fill-induct [consumes 1, case-names MHole MVar MFun]*:

assumes $t =_f (C, ts)$

and $\bigwedge t. P \ t \ \text{MHole } [t]$

and $\bigwedge x. P \ (\text{Var } x) \ (\text{MVar } x) \ []$

and $\bigwedge f \ ss \ Cs \ ts. \llbracket \text{length } Cs = \text{length } ss; \text{sum-list } (\text{map num-holes } Cs) = \text{length}$

$ts;$

$\forall i < \text{length } ss. ss ! i =_f (Cs ! i, \text{partition-holes } ts \ Cs ! i) \wedge$

$P \ (ss ! i) \ (Cs ! i) \ (\text{partition-holes } ts \ Cs ! i) \rrbracket$

$\implies P \ (\text{Fun } f \ ss) \ (\text{MFun } f \ Cs) \ ts$

shows $P \ t \ C \ ts$

$\langle \text{proof} \rangle$

lemma *hole-poss-subset-poss*:

assumes $s =_f (C, ss)$

shows $\text{hole-poss } C \subseteq \text{poss } s$

$\langle \text{proof} \rangle$

fun *hole-num*

where

$\text{hole-num } [] \text{ MHole} = 0 \mid$

$\text{hole-num } (i \# q) (\text{MFun } f \text{ Cs}) = \text{sum-list } (\text{map num-holes } (\text{take } i \text{ Cs})) + \text{hole-num } q \text{ (Cs ! } i)$

lemma *hole-poss-nth-subt-at*:

assumes $t =_f (C, ts)$ **and** $p \in \text{hole-poss } C$

shows $\text{hole-num } p \text{ C} < \text{length } ts \wedge t \mid - p = ts ! \text{hole-num } p \text{ C}$

$\langle \text{proof} \rangle$

lemma *eqf-Fun-MFun*:

assumes $\text{Fun } f \text{ ss} =_f (\text{MFun } g \text{ Cs}, ts)$

shows $g = f \wedge \text{length } Cs = \text{length } ss \wedge \text{sum-list } (\text{map num-holes } Cs) = \text{length } ts \wedge$

$(\forall i < \text{length } ss. ss ! i =_f (Cs ! i, \text{partition-holes } ts \text{ Cs ! } i))$

$\langle \text{proof} \rangle$

lemma *fill-holes-eq-Var-cases*:

assumes $\text{num-holes } C = \text{length } ts$

and $\text{fill-holes } C \text{ ts} = \text{Var } x$

obtains $C = \text{MHole} \wedge ts = [\text{Var } x] \mid C = \text{MVar } x \wedge ts = []$

$\langle \text{proof} \rangle$

lemma *num-holes-inf-mctxt-le*:

assumes $s =_f (C, ts)$ **and** $s =_f (D, us)$

shows $\text{num-holes } (C \sqcap D) \leq \text{num-holes } C + \text{num-holes } D$

$\langle \text{proof} \rangle$

lemma *map-inf-mctxt-zip-mctxt-of-term [simp]*:

$\text{map } (\lambda(x, y). x \sqcap y) (\text{zip } (\text{map mctxt-of-term } ts) (\text{map mctxt-of-term } ts)) = \text{map mctxt-of-term } ts$

$\langle \text{proof} \rangle$

lemma *inf-mctxt-ctxt-apply-term [simp]*:

$\text{mctxt-of-term } (C \langle t \rangle) \sqcap \text{mctxt-of-ctxt } C = \text{mctxt-of-ctxt } C$

$\text{mctxt-of-ctxt } C \sqcap \text{mctxt-of-term } (C \langle t \rangle) = \text{mctxt-of-ctxt } C$

$\langle \text{proof} \rangle$

lemma *inf-fill-holes-mctxt-MHoles*:

$\text{num-holes } C = \text{length } Cs \implies \text{length } Ds = \text{length } Cs \implies$

$\forall i < \text{length } Cs. Cs ! i = \text{MHole} \vee Ds ! i = \text{MHole} \implies$

$\text{fill-holes-mctxt } C \text{ } Cs \sqcap \text{fill-holes-mctxt } C \text{ } Ds = C$
 $\langle \text{proof} \rangle$

lemma *inf-fill-holes-mctxt-two-MHoles* [simp]: $\text{num-holes } C = 2 \implies$
 $\text{fill-holes-mctxt } C \text{ } [MHole, D] \sqcap \text{fill-holes-mctxt } C \text{ } [E, MHole] = C$
 $\langle \text{proof} \rangle$

lemma *two-subterms-cases*:

assumes $s = C\langle t \rangle$ **and** $s = D\langle u \rangle$
obtains $(eq) \ C = D$ **and** $t = u$
 $| \text{ (nested1) } C' \text{ where } C' \neq \square \text{ and } C = D \circ_c C'$
 $| \text{ (nested2) } D' \text{ where } D' \neq \square \text{ and } D = C \circ_c D'$
 $| \text{ (parallel1) } E \text{ where } \text{num-holes } E = 2$
 $\text{and mctxt-of-ctxt } C = \text{fill-holes-mctxt } E \text{ } [MHole, \text{mctxt-of-term } u]$
 $\text{and mctxt-of-ctxt } D = \text{fill-holes-mctxt } E \text{ } [\text{mctxt-of-term } t, MHole]$
 $| \text{ (parallel2) } E \text{ where } \text{num-holes } E = 2$
 $\text{and mctxt-of-ctxt } C = \text{fill-holes-mctxt } E \text{ } [\text{mctxt-of-term } u, MHole]$
 $\text{and mctxt-of-ctxt } D = \text{fill-holes-mctxt } E \text{ } [MHole, \text{mctxt-of-term } t]$
 $\langle \text{proof} \rangle$

lemma *two-hole-ctxt-inf-conv*:

$\text{num-holes } E = 2 \implies$
 $\text{mctxt-of-ctxt } C = \text{fill-holes-mctxt } E \text{ } [MHole, \text{mctxt-of-term } u] \implies$
 $\text{mctxt-of-ctxt } D = \text{fill-holes-mctxt } E \text{ } [\text{mctxt-of-term } t, MHole] \implies$
 $\text{mctxt-of-ctxt } C \sqcap \text{mctxt-of-ctxt } D = E$
 $\langle \text{proof} \rangle$

lemma *map-length-take-partition-by*:

$i < \text{length } ys \implies \text{sum-list } ys = \text{length } xs \implies$
 $\text{map length } (\text{take } i \text{ } (\text{partition-by } xs \text{ } ys)) = \text{take } i \text{ } ys$
 $\langle \text{proof} \rangle$

Closure under contexts can be lifted to multihole contexts.

lemma *ctxt-imp-mctxt*:

assumes $\forall s \ t \ C. (s, t) \in R \longrightarrow (C\langle s \rangle, C\langle t \rangle) \in R$
and $(t, u) \in R$
and $\text{num-holes } C = \text{length } ss_1 + \text{length } ss_2 + 1$
shows $(\text{fill-holes } C \text{ } (ss_1 @ t \# ss_2), \text{fill-holes } C \text{ } (ss_1 @ u \# ss_2)) \in R$
 $\langle \text{proof} \rangle$

lemma *mctxt-of-term-fill-holes'*:

$\text{num-holes } C = \text{length } ts \implies \text{mctxt-of-term } (\text{fill-holes } C \text{ } ts) = \text{fill-holes-mctxt } C$
 $(\text{map mctxt-of-term } ts)$
 $\langle \text{proof} \rangle$

lemma *vars-term-fill-holes'*:

$\text{num-holes } C = \text{length } ts \implies \text{vars-term } (\text{fill-holes } C \text{ } ts) = \bigcup (\text{vars-term } \text{ ` } \text{set } ts)$
 $\cup \text{vars-mctxt } C$
 $\langle \text{proof} \rangle$

lemma *vars-mctx-linear*: **assumes** $t =_f (C, ts)$
 linear-term t
shows $\text{vars-mctx } C \cap \bigcup (\text{vars-term } \text{' set } ts) = \{\}$
 $\langle \text{proof} \rangle$

lemma *mctx-of-term-var-subst*:
 $\text{mctx-of-term } (t \cdot (\text{Var} \circ f)) = \text{map-vars-mctx } f (\text{mctx-of-term } t)$
 $\langle \text{proof} \rangle$

lemma *subst-apply-mctx-map-vars-mctx-conv*:
 $C \cdot \text{mc } (\text{Var} \circ f) = \text{map-vars-mctx } f C$
 $\langle \text{proof} \rangle$

lemma *map-vars-mctx-mono*:
 $C \leq D \implies \text{map-vars-mctx } f C \leq \text{map-vars-mctx } f D$
 $\langle \text{proof} \rangle$

lemma *map-vars-mctx-less-eq-decomp*:
 assumes $C \leq \text{map-vars-mctx } f D$
 obtains C' **where** $\text{map-vars-mctx } f C' = C \text{ } C' \leq D$
 $\langle \text{proof} \rangle$

6.4 All positions of a multi-hole context

fun *all-poss-mctx* :: $(f, v) \text{ mctx} \Rightarrow \text{pos set}$
where
 $\text{all-poss-mctx } (MVar x) = \{\square\}$
 $\text{all-poss-mctx } MHole = \{\square\}$
 $\text{all-poss-mctx } (MFun f cs) = \{\square\} \cup \bigcup (\text{set } (\text{map } (\lambda i. (\lambda p. i \# p) \text{' all-poss-mctx } (cs ! i)) [0 ..< \text{length } cs]))$

lemma *all-poss-mctx-simp* [*simp*]:
 $\text{all-poss-mctx } (MFun f cs) = \{\square\} \cup \{i \# p \mid i p. i < \text{length } cs \wedge p \in \text{all-poss-mctx } (cs ! i)\}$
 $\langle \text{proof} \rangle$

declare *all-poss-mctx.simps*(\mathcal{J})[*simp del*]

lemma *all-poss-mctx-conv*:
 $\text{all-poss-mctx } C = \text{poss-mctx } C \cup \text{hole-poss } C$
 $\langle \text{proof} \rangle$

lemma *root-in-all-poss-mctx*[*simp*]:
 $\square \in \text{all-poss-mctx } C$
 $\langle \text{proof} \rangle$

lemma *hole-poss-mctx-of-term*[*simp*]:
 $\text{hole-poss } (\text{mctx-of-term } t) = \{\}$

$\langle \text{proof} \rangle$

lemma *poss-mctxt-mctxt-of-term*[simp]:
 $\text{poss-mctxt } (\text{mctxt-of-term } t) = \text{poss } t$
 $\langle \text{proof} \rangle$

lemma *hole-poss-subst*: $\text{hole-poss } (C \cdot \text{mc } \sigma) = \text{hole-poss } C$
 $\langle \text{proof} \rangle$

lemma *all-poss-mctxt-mctxt-of-term*[simp]:
 $\text{all-poss-mctxt } (\text{mctxt-of-term } t) = \text{poss } t$
 $\langle \text{proof} \rangle$

lemma *mctxt-of-term-leq-imp-eq*:
 $\text{mctxt-of-term } t \leq C \iff \text{mctxt-of-term } t = C$
 $\langle \text{proof} \rangle$

lemma *mctxt-of-term-inj*:
 $\text{mctxt-of-term } s = \text{mctxt-of-term } t \iff s = t$
 $\langle \text{proof} \rangle$

lemma *all-poss-mctxt-map-vars-mctxt* [simp]:
 $\text{all-poss-mctxt } (\text{map-vars-mctxt } f \ C) = \text{all-poss-mctxt } C$
 $\langle \text{proof} \rangle$

lemma *fill-holes-mctxt-extends-all-poss*:
assumes $\text{length } Ds = \text{num-holes } C$ **shows** $\text{all-poss-mctxt } C \subseteq \text{all-poss-mctxt } (\text{fill-holes-mctxt } C \ Ds)$
 $\langle \text{proof} \rangle$

lemma *eqF-substD*:
assumes $t \cdot \sigma =_f (C, ss)$
 $\text{hole-poss } C \subseteq \text{poss } t$
shows $\exists \ D \ ts. t =_f (D, ts) \wedge C = D \cdot \text{mc } \sigma \wedge ss = \text{map } (\lambda \ ti. ti \cdot \sigma) \ ts$
 $\langle \text{proof} \rangle$

6.5 More operations on multi-hole contexts

fun *root-mctxt* :: $('f, 'v) \text{ mctxt} \Rightarrow ('f \times \text{nat}) \text{ option}$ **where**
 $\text{root-mctxt } \text{MHole} = \text{None}$
 $| \text{root-mctxt } (\text{MVar } x) = \text{None}$
 $| \text{root-mctxt } (\text{MFun } f \ Cs) = \text{Some } (f, \text{length } Cs)$

fun *mreplace-at* :: $('f, 'v) \text{ mctxt} \Rightarrow \text{pos} \Rightarrow ('f, 'v) \text{ mctxt} \Rightarrow ('f, 'v) \text{ mctxt}$ **where**
 $\text{mreplace-at } C \ [] \ D = D$
 $| \text{mreplace-at } (\text{MFun } f \ Cs) \ (i \ \# \ p) \ D = \text{MFun } f \ (\text{take } i \ Cs @ \text{mreplace-at } (Cs \ ! \ i) \ p \ D \ \# \ \text{drop } (i+1) \ Cs)$

fun *subm-at* :: ('f, 'v) mctxt \Rightarrow pos \Rightarrow ('f, 'v) mctxt **where**
 subm-at *C* [] = *C*
 | *subm-at* (MFun *f* *Cs*) (*i* # *p*) = *subm-at* (*Cs* ! *i*) *p*

lemma *subm-at-hole-poss[simp]*:
p \in *hole-poss* *C* \implies *subm-at* *C* *p* = *MHole*
 <proof>

lemma *subm-at-mctxt-of-term*:
p \in *poss* *t* \implies *subm-at* (*mctxt-of-term* *t*) *p* = *mctxt-of-term* (*subt-at* *t* *p*)
 <proof>

lemma *subm-at-mreplace-at[simp]*:
p \in *all-poss-mctxt* *C* \implies *subm-at* (*mreplace-at* *C* *p* *D*) *p* = *D*
 <proof>

lemma *replace-at-subm-at[simp]*:
p \in *all-poss-mctxt* *C* \implies *mreplace-at* *C* *p* (*subm-at* *C* *p*) = *C*
 <proof>

lemma *all-poss-mctxt-mreplace-atI1*:
p \in *all-poss-mctxt* *C* \implies *q* \in *all-poss-mctxt* *C* \implies \neg (*p* <_{*p*} *q*) \implies *q* \in *all-poss-mctxt*
 (*mreplace-at* *C* *p* *D*)
 <proof>

lemma *funas-mctxt-sup-mctxt*:
 (*C*, *D*) \in *comp-mctxt* \implies *funas-mctxt* (*C* \sqcup *D*) = *funas-mctxt* *C* \cup *funas-mctxt* *D*
 <proof>

lemma *mctxt-of-term-not-hole [simp]*:
mctxt-of-term *t* \neq *MHole*
 <proof>

lemma *funas-mctxt-mctxt-of-term [simp]*:
funas-mctxt (*mctxt-of-term* *t*) = *funas-term* *t*
 <proof>

lemma *funas-mctxt-mreplace-at*:
assumes *p* \in *all-poss-mctxt* *C*
shows *funas-mctxt* (*mreplace-at* *C* *p* *D*) \subseteq *funas-mctxt* *C* \cup *funas-mctxt* *D*
 <proof>

lemma *funas-mctxt-mreplace-at-hole*:
assumes *p* \in *hole-poss* *C*
shows *funas-mctxt* (*mreplace-at* *C* *p* *D*) = *funas-mctxt* *C* \cup *funas-mctxt* *D* (**is**
 ?*L* = ?*R*)

$\langle \text{proof} \rangle$

lemma *map-vars-mctxt-fill-holes-mctxt*:

assumes *num-holes* $C = \text{length } Cs$

shows $\text{map-vars-mctxt } f (\text{fill-holes-mctxt } C \ Cs) = \text{fill-holes-mctxt } (\text{map-vars-mctxt } f \ C) (\text{map } (\text{map-vars-mctxt } f) \ Cs)$

$\langle \text{proof} \rangle$

lemma *map-vars-mctxt-map-vars-mctxt[simp]*:

shows $\text{map-vars-mctxt } f (\text{map-vars-mctxt } g \ C) = \text{map-vars-mctxt } (f \circ g) \ C$

$\langle \text{proof} \rangle$

lemma *funas-mctxt-fill-holes*:

assumes *num-holes* $C = \text{length } ts$

shows $\text{funas-term } (\text{fill-holes } C \ ts) = \text{funas-mctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-term } ts))$

$\langle \text{proof} \rangle$

lemma *mctxt-neq-mholeE*:

$x \neq \text{MHole} \implies (\bigwedge v. x = \text{MVar } v \implies P) \implies (\bigwedge f \ Cs. x = \text{MFun } f \ Cs \implies P) \implies P$

$\langle \text{proof} \rangle$

lemma *prefix-comp-mctxt*:

$C \leq E \implies D \leq E \implies (C, D) \in \text{comp-mctxt}$

$\langle \text{proof} \rangle$

lemma *less-eq-mctxt-sup-conv1*:

$(C, D) \in \text{comp-mctxt} \implies C \leq D \longleftrightarrow C \sqcup D = D$

$\langle \text{proof} \rangle$

lemma *less-eq-mctxt-sup-conv2*:

$(C, D) \in \text{comp-mctxt} \implies D \leq C \longleftrightarrow C \sqcup D = C$

$\langle \text{proof} \rangle$

lemma *comp-mctxt-mctxt-of-term1[dest!]*:

$(C, \text{mctxt-of-term } t) \in \text{comp-mctxt} \implies C \leq \text{mctxt-of-term } t$

$\langle \text{proof} \rangle$

lemmas $\text{comp-mctxt-mctxt-of-term2[dest!]} = \text{comp-mctxt-mctxt-of-term1[OF comp-mctxt-sym]}$

lemma *mfun-leq-mfunI*:

$f = g \implies \text{length } Cs = \text{length } Ds \implies (\bigwedge i. i < \text{length } Ds \implies Cs ! i \leq Ds ! i) \implies \text{MFun } f \ Cs \leq \text{MFun } g \ Ds$

$\langle \text{proof} \rangle$

lemma *prefix-mctxt-sup*:

assumes $C \leq (E :: ('f, 'v) \text{mctxt}) \ D \leq E$ **shows** $C \sqcup D \leq E$

$\langle \text{proof} \rangle$

lemma *mreplace-at-leqI*:

$p \in \text{all-poss-mctxt } C \implies C \leq E \implies D \leq \text{subm-at } E \ p \implies \text{mreplace-at } C \ p \ D \leq E$
 $\langle \text{proof} \rangle$

lemma *prefix-and-fewer-holes-implies-equal-mctxt*:

$C \leq D \implies \text{hole-poss } C \subseteq \text{hole-poss } D \implies C = D$
 $\langle \text{proof} \rangle$

lemma *compare-mreplace-at*:

$p \in \text{poss-mctxt } C \implies \text{mreplace-at } C \ p \ D \leq \text{mreplace-at } C \ p \ E \longleftrightarrow D \leq E$
 $\langle \text{proof} \rangle$

lemma *merge-mreplace-at*:

$p \in \text{poss-mctxt } C \implies \text{mreplace-at } C \ p \ (D \sqcup E) = \text{mreplace-at } C \ p \ D \sqcup \text{mreplace-at } C \ p \ E$
 $\langle \text{proof} \rangle$

lemma *compare-mreplace-atI'*:

$C \leq D \implies C' \leq D' \implies p \in \text{all-poss-mctxt } C \implies \text{mreplace-at } C \ p \ C' \leq \text{mreplace-at } D \ p \ D'$
 $\langle \text{proof} \rangle$

lemma *compare-mreplace-atI*:

$C \leq D \implies C' \leq D' \implies p \in \text{poss-mctxt } C \implies \text{mreplace-at } C \ p \ C' \leq \text{mreplace-at } D \ p \ D'$
 $\langle \text{proof} \rangle$

lemma *all-poss-mctxt-mono*:

$C \leq D \implies \text{all-poss-mctxt } C \subseteq \text{all-poss-mctxt } D$
 $\langle \text{proof} \rangle$

lemma *all-poss-mctxt-inf-mctxt*:

$(C, D) \in \text{comp-mctxt} \implies \text{all-poss-mctxt } (C \sqcap D) = \text{all-poss-mctxt } C \cap \text{all-poss-mctxt } D$
 $\langle \text{proof} \rangle$

lemma *less-eq-subm-at*:

$p \in \text{all-poss-mctxt } C \implies C \leq D \implies \text{subm-at } C \ p \leq \text{subm-at } D \ p$
 $\langle \text{proof} \rangle$

lemma *inf-subm-at*:

$p \in \text{all-poss-mctxt } (C \sqcap D) \implies \text{subm-at } (C \sqcap D) \ p = \text{subm-at } C \ p \sqcap \text{subm-at } D \ p$
 $\langle \text{proof} \rangle$

lemma *less-eq-fill-holesI*:

assumes $\text{length } Ds = \text{num-holes } C \text{ length } Es = \text{num-holes } C$

$\bigwedge i. i < \text{num-holes } C \implies Ds ! i \leq Es ! i$

shows $\text{fill-holes-mctxt } C Ds \leq \text{fill-holes-mctxt } C Es$

<proof>

lemma *less-eq-fill-holesD*:

assumes $\text{length } Ds = \text{num-holes } C \text{ length } Es = \text{num-holes } C$

$\text{fill-holes-mctxt } C Ds \leq \text{fill-holes-mctxt } C Es \implies i < \text{num-holes } C$

shows $Ds ! i \leq Es ! i$

<proof>

lemma *less-eq-fill-holes-iff*:

assumes $\text{length } Ds = \text{num-holes } C \text{ length } Es = \text{num-holes } C$

shows $\text{fill-holes-mctxt } C Ds \leq \text{fill-holes-mctxt } C Es \iff (\forall i < \text{num-holes } C. Ds ! i \leq Es ! i)$

<proof>

lemma *fill-holes-mctxt-suffix[simp]*:

assumes $\text{length } Ds = \text{num-holes } C$ **shows** $C \leq \text{fill-holes-mctxt } C Ds$

<proof>

lemma *fill-holes-mctxt-id*:

assumes $\text{length } Ds = \text{num-holes } C$ **shows** $C = \text{fill-holes-mctxt } C Ds$ **shows** $\text{set } Ds \subseteq \{MHole\}$

<proof>

lemma *fill-holes-suffix[simp]*:

$\text{num-holes } C = \text{length } ts \implies C \leq \text{mctxt-of-term } (\text{fill-holes } C ts)$

<proof>

6.6 An inverse of fill-holes

fun *unfill-holes* :: $(f, 'v) \text{ mctxt} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term list}$ **where**

unfill-holes *MHole* *t* = [*t*]

| *unfill-holes* (*MVar* *w*) (*Var* *v*) = (if *v* = *w* then [] else undefined)

| *unfill-holes* (*MFun* *g* *Cs*) (*Fun* *f* *ts*) = (if *f* = *g* \wedge $\text{length } ts = \text{length } Cs$ then $\text{concat } (\text{map } (\lambda i. \text{unfill-holes } (Cs ! i) (ts ! i)) [0..<\text{length } ts])$ else undefined)

lemma *length-unfill-holes[simp]*:

assumes $C \leq \text{mctxt-of-term } t$

shows $\text{length } (\text{unfill-holes } C t) = \text{num-holes } C$

<proof>

lemma *fill-unfill-holes*:

assumes $C \leq \text{mctxt-of-term } t$

shows $\text{fill-holes } C (\text{unfill-holes } C t) = t$

<proof>

lemma *unfill-fill-holes*:

assumes $\text{length } ts = \text{num-holes } C$

shows $\text{unfill-holes } C (\text{fill-holes } C ts) = ts$

$\langle \text{proof} \rangle$

lemma *unfill-holes-subt*:

assumes $C \leq \text{mctxt-of-term } t$ **and** $t' \in \text{set } (\text{unfill-holes } C t)$

shows $t' \sqsubseteq t$

$\langle \text{proof} \rangle$

lemma *factor-hole-pos-by-prefix*:

assumes $C \leq D$ $p \in \text{hole-poss } D$

obtains q **where** $q \leq_p p$ $q \in \text{hole-poss } C$

$\langle \text{proof} \rangle$

lemma *concat-map-zip-upt*: **assumes** $\bigwedge i. i < n \implies \text{length } (f i) = \text{length } (g i)$

shows $\text{concat } (\text{map } (\lambda i. \text{zip } (f i) (g i)) [0..<n]) = \text{zip } (\text{concat } (\text{map } f [0..<n]))$
 $(\text{concat } (\text{map } g [0..<n]))$

$\langle \text{proof} \rangle$

lemma *unfill-holes-by-prefix'*:

assumes $\text{num-holes } C = \text{length } Ds$ $\text{fill-holes-mctxt } C Ds \leq \text{mctxt-of-term } t$

shows $\text{unfill-holes } (\text{fill-holes-mctxt } C Ds) t = \text{concat } (\text{map } (\lambda(D, t). \text{unfill-holes } D t) (\text{zip } Ds (\text{unfill-holes } C t)))$

$\langle \text{proof} \rangle$

lemma *unfill-holes-var-subst*:

$C \leq \text{mctxt-of-term } t \implies \text{unfill-holes } (\text{map-vars-mctxt } f C) (t \cdot (\text{Var} \circ f)) = \text{map}$
 $(\lambda t. t \cdot (\text{Var} \circ f)) (\text{unfill-holes } C t)$

$\langle \text{proof} \rangle$

6.7 Ditto for *fill-holes-mctxt*

fun *unfill-holes-mctxt* :: $(f, 'v) \text{ mctxt} \Rightarrow (f, 'v) \text{ mctxt} \Rightarrow (f, 'v) \text{ mctxt list}$ **where**

$\text{unfill-holes-mctxt } \text{MHole } D = [D]$

| $\text{unfill-holes-mctxt } (\text{MVar } w) (\text{MVar } v) = (\text{if } v = w \text{ then } [] \text{ else undefined})$

| $\text{unfill-holes-mctxt } (\text{MFun } g Cs) (\text{MFun } f Ds) = (\text{if } f = g \wedge \text{length } Ds = \text{length } Cs$
 then

$\text{concat } (\text{map } (\lambda i. \text{unfill-holes-mctxt } (Cs ! i) (Ds ! i)) [0..<\text{length } Ds]) \text{ else undefined})$

lemma *length-unfill-holes-mctxt* [simp]:

assumes $C \leq D$

shows $\text{length } (\text{unfill-holes-mctxt } C D) = \text{num-holes } C$

$\langle \text{proof} \rangle$

lemma *fill-unfill-holes-mctxt*:

assumes $C \leq D$

shows $\text{fill-holes-mctxt } C (\text{unfill-holes-mctxt } C D) = D$

$\langle \text{proof} \rangle$

lemma *unfill-fill-holes-mctxt*:

assumes $\text{length } Ds = \text{num-holes } C$

shows $\text{unfill-holes-mctxt } C (\text{fill-holes-mctxt } C Ds) = Ds$

$\langle \text{proof} \rangle$

lemma *unfill-holes-mctxt-mctxt-of-term*:

assumes $C \leq \text{mctxt-of-term } t$

shows $\text{unfill-holes-mctxt } C (\text{mctxt-of-term } t) = \text{map mctxt-of-term } (\text{unfill-holes } C t)$

$\langle \text{proof} \rangle$

6.8 Function symbols of prefixes

lemma *funas-prefix[simp]*:

$C \leq D \implies \text{fn} \in \text{funas-mctxt } C \implies \text{fn} \in \text{funas-mctxt } D$

$\langle \text{proof} \rangle$

6.9 Parallel Rewriting using Multihole Contexts

datatype $(f, v)\text{par-info} = \text{Par-Info}$

$(\text{par-left}: (f, v)\text{term})$

$(\text{par-right}: (f, v)\text{term})$

$(\text{par-rule}: (f, v)\text{rule})$

abbreviation *par-lefts* **where** $\text{par-lefts} \equiv \text{map par-left}$

abbreviation *par-rights* **where** $\text{par-rights} \equiv \text{map par-right}$

abbreviation *par-rules* **where** $\text{par-rules} \equiv (\lambda \text{ info. par-rule } \text{' set info})$

definition *par-cond* :: $(f, v)\text{trs} \Rightarrow (f, v)\text{par-info} \Rightarrow \text{bool}$ **where**

$\text{par-cond } R \text{ info} = (\text{par-rule info} \in R \wedge (\text{par-left info}, \text{par-right info}) \in \text{rrstep } \{\text{par-rule info}\})$

abbreviation *par-conds* **where** $\text{par-conds } R \equiv \lambda \text{ infos. Ball } (\text{set infos}) (\text{par-cond } R)$

lemma *par-cond-imp-rrstep*: **assumes** $\text{par-cond } R \text{ info}$

shows $(\text{par-left info}, \text{par-right info}) \in \text{rrstep } R$

$\langle \text{proof} \rangle$

lemma *par-conds-imp-rrstep*: **assumes** $\text{par-conds } R \text{ infos}$

and $s = \text{par-lefts infos} ! i \text{ } t = \text{par-rights infos} ! i$

and $i < \text{length infos}$

shows $(s, t) \in \text{rrstep } R$

$\langle \text{proof} \rangle$

lemma *par-rstep-mctxt-rrstepI* :

assumes $s =_f (C, ss)$ **and** $t =_f (C, ts)$

and $\forall i < \text{length } ss. (ss ! i, ts ! i) \in \text{rrstep } R$

shows $(s, t) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

definition par-rstep-mctxt **where**

$\text{par-rstep-mctxt } R \ C \ \text{infos} = \{(s, t). s =_f (C, \text{par-lefts infos}) \wedge t =_f (C, \text{par-rights infos}) \wedge \text{par-conds } R \ \text{infos}\}$

lemma par-rstep-mctxtI : **assumes** $s =_f (C, \text{par-lefts infos}) \ t =_f (C, \text{par-rights infos})$ $\text{par-conds } R \ \text{infos}$

shows $(s, t) \in \text{par-rstep-mctxt } R \ C \ \text{infos}$
 $\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-reflI}$: $(s, s) \in \text{par-rstep-mctxt } R \ (\text{mctxt-of-term } s) \ \square$
 $\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-varI}$: $(\text{Var } x, \text{Var } x) \in \text{par-rstep-mctxt } R \ (\text{MVar } x) \ \square$
 $\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-MHoleI}$: $(l, r) \in R \implies s = l \cdot \sigma \implies t = r \cdot \sigma \implies \text{infos} = [\text{Par-Info } s \ t \ (l, r)]$
 $\implies (s, t) \in \text{par-rstep-mctxt } R \ \text{MHole infos}$
 $\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-funI}$:

assumes $\text{rec: } \bigwedge i. i < \text{length } ts \implies (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R \ (Cs ! i)$
 $(\text{infos} ! i)$

and $\text{len: } \text{length } ss = \text{length } ts \ \text{length } Cs = \text{length } ts \ \text{length } \text{infos} = \text{length } ts$

shows $(\text{Fun } f \ ss, \text{Fun } f \ ts) \in \text{par-rstep-mctxt } R \ (\text{MFun } f \ Cs) \ (\text{concat infos})$
 $\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-funI-ex}$:

assumes $\bigwedge i. i < \text{length } ts \implies \exists \ C \ \text{infos}. (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R \ C \ \text{infos}$

and $\text{length } ss = \text{length } ts$

shows $\exists \ C \ \text{infos}. (\text{Fun } f \ ss, \text{Fun } f \ ts) \in \text{par-rstep-mctxt } R \ C \ \text{infos} \wedge C \neq \text{MHole}$

$\langle \text{proof} \rangle$

lemma $\text{par-rstep-mctxt-mono}$: **assumes** $R \subseteq S$

shows $\text{par-rstep-mctxt } R \ C \ \text{infos} \subseteq \text{par-rstep-mctxt } S \ C \ \text{infos}$
 $\langle \text{proof} \rangle$

lemma par-rstep-mctxtE :

assumes $(s, t) \in \text{par-rstep } R$

obtains $C \ \text{infos}$ **where** $s =_f (C, \text{par-lefts infos})$ **and** $t =_f (C, \text{par-rights infos})$

and $\text{par-conds } R \ \text{infos}$

$\langle \text{proof} \rangle$

lemma *par-rstep-par-rstep-mctxt-conv*:

$(s, t) \in \text{par-rstep } R \iff (\exists C \text{ infos. } (s, t) \in \text{par-rstep-mctxt } R \ C \ \text{infos})$
 $\langle \text{proof} \rangle$

fun *subst-apply-par-info* :: $(f, v) \text{par-info} \Rightarrow (f, v) \text{subst} \Rightarrow (f, v) \text{par-info}$ (**infixl**
 $\cdot \text{pi}$ 67) **where**
 $\text{Par-Info } s \ t \ r \cdot \text{pi } \sigma = \text{Par-Info } (s \cdot \sigma) \ (t \cdot \sigma) \ r$

lemma *subst-apply-par-info-simps*[*simp*]:

$\text{par-left } (\text{info} \cdot \text{pi } \sigma) = \text{par-left } \text{info} \cdot \sigma$
 $\text{par-right } (\text{info} \cdot \text{pi } \sigma) = \text{par-right } \text{info} \cdot \sigma$
 $\text{par-rule } (\text{info} \cdot \text{pi } \sigma) = \text{par-rule } \text{info}$
 $\text{par-cond } R \ \text{info} \implies \text{par-cond } R \ (\text{info} \cdot \text{pi } \sigma)$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mctxt-subst*: **assumes** $(s, t) \in \text{par-rstep-mctxt } R \ C \ \text{infos}$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-mctxt } R \ (C \cdot \text{mc } \sigma) \ (\text{map } (\lambda i. i \cdot \text{pi } \sigma) \ \text{infos})$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mctxt-MVarE*:

assumes $(s, t) \in \text{par-rstep-mctxt } R \ (MVar \ x) \ \text{infos}$
shows $s = Var \ x \ t = Var \ x \ \text{infos} = []$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mctxt-MHoleE*:

assumes $(s, t) \in \text{par-rstep-mctxt } R \ MHole \ \text{infos}$
obtains *info* **where**
 $\text{par-left } \text{info} = s$
 $\text{par-right } \text{info} = t$
 $\text{infos} = [\text{info}]$
 $(s, t) \in \text{rrstep } R$
 $\text{par-cond } R \ \text{info}$
 $\langle \text{proof} \rangle$

lemma *par-rstep-mctxt-MFunD*:

assumes $(s, t) \in \text{par-rstep-mctxt } R \ (MFun \ f \ Cs) \ \text{infos}$
shows $\exists \ ss \ ts \ \text{Infos.}$
 $s = Fun \ f \ ss \wedge$
 $t = Fun \ f \ ts \wedge$
 $\text{length } ss = \text{length } Cs \wedge$
 $\text{length } ts = \text{length } Cs \wedge$
 $\text{length } \text{Infos} = \text{length } Cs \wedge$
 $\text{infos} = \text{concat } \text{Infos} \wedge$
 $(\forall i < \text{length } Cs. (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R \ (Cs ! i) \ (\text{Infos} ! i))$
 $\langle \text{proof} \rangle$

end

7 Multi-Step Rewriting

```

theory Multistep
imports Trs
begin

```

Multi-step rewriting (without proof terms).

```

inductive-set
  mstep :: ('f, 'v) trs  $\Rightarrow$  ('f, 'v) term rel
  for R
where
  Var: (Var x, Var x)  $\in$  mstep R |
  args:  $\bigwedge f n ss ts. \llbracket \text{length } ss = n; \text{length } ts = n; \forall i < n. (ss ! i, ts ! i) \in \text{mstep } R \rrbracket \implies$ 
    (Fun f ss, Fun f ts)  $\in$  mstep R |
  rule:  $\bigwedge l r \sigma \tau. \llbracket (l, r) \in R; \forall x \in \text{vars-term } l. (\sigma x, \tau x) \in \text{mstep } R \rrbracket \implies$ 
    (l  $\cdot$   $\sigma$ , r  $\cdot$   $\tau$ )  $\in$  mstep R

```

```

lemma mstep-refl [simp]:
  (t, t)  $\in$  mstep R
  <proof>

```

```

lemma mstep-ctxt:
  assumes (s, t)  $\in$  mstep R
  shows (C<s>, C<t>)  $\in$  mstep R
  <proof>

```

```

lemma rstep-imp-mstep:
  assumes (s, t)  $\in$  rstep R
  shows (s, t)  $\in$  mstep R
  <proof>

```

```

lemma rstep-mstep-subset:
  rstep R  $\subseteq$  mstep R
  <proof>

```

```

lemma subst-rsteps-imp-rule-rsteps:
  assumes  $\forall x \in \text{vars-term } l. (\sigma x, \tau x) \in (\text{rstep } R)^*$ 
  and (l, r)  $\in$  R
  shows (l  $\cdot$   $\sigma$ , r  $\cdot$   $\tau$ )  $\in$  (rstep R)*
  <proof>

```

```

lemma mstep-imp-rsteps:
  assumes (s, t)  $\in$  mstep R
  shows (s, t)  $\in$  (rstep R)*
  <proof>

```

```

lemma mstep-rsteps-subset:
  shows mstep R  $\subseteq$  (rstep R)*

```

<proof>

lemma *mstep-mono*: $R \subseteq S \implies \text{mstep } R \subseteq \text{mstep } S$
<proof>

Thus if *mstep* *R* has the diamond property, then *rstep* *R* is confluent.

lemma *Var-mstep*:
assumes *: $\bigwedge l\ r. (l, r) \in R \implies \neg \text{is-Var } l$
and $(\text{Var } x, t) \in \text{mstep } R$
shows $t = \text{Var } x$
<proof>

Maximal multi-step rewriting.

inductive-set
 $\text{mmstep} :: ('f, 'v) \text{trs} \Rightarrow ('f, 'v) \text{term rel}$
for *R*
where
 $\text{Var}: (\text{Var } x, \text{Var } x) \in \text{mmstep } R \mid$
 $\text{args}: \bigwedge f\ n\ ss\ ts. \llbracket \text{length } ss = n; \text{length } ts = n;$
 $\neg (\exists (l, r) \in R. \exists \sigma. \text{Fun } f\ ss = l \cdot \sigma);$
 $\forall i < n. (ss ! i, ts ! i) \in \text{mmstep } R \rrbracket \implies$
 $(\text{Fun } f\ ss, \text{Fun } f\ ts) \in \text{mmstep } R \mid$
 $\text{rule}: \bigwedge l\ r\ \sigma\ \tau. \llbracket (l, r) \in R; \forall x \in \text{vars-term } l. (\sigma\ x, \tau\ x) \in \text{mmstep } R \rrbracket \implies$
 $(l \cdot \sigma, r \cdot \tau) \in \text{mmstep } R$

lemma *mmstep-imp-mstep*:
assumes $(s, t) \in \text{mmstep } R$
shows $(s, t) \in \text{mstep } R$
<proof>

lemma *mmstep-mstep-subset*:
 $\text{mmstep } R \subseteq \text{mstep } R$
<proof>

end

theory *Option-Util*
imports *Main*
begin

primrec *option-to-list* :: $'a \text{ option} \Rightarrow 'a \text{ list}$
where
 $\text{option-to-list } (\text{Some } a) = [a] \mid$
 $\text{option-to-list } \text{None} = []$

lemma *set-option-to-list-sound* [*simp*]:
 $\text{set } (\text{option-to-list } t) = \text{set-option } t$
<proof>

```

fun fun-of-map :: ('a  $\Rightarrow$  'b option)  $\Rightarrow$  'b  $\Rightarrow$  ('a  $\Rightarrow$  'b) where
  fun-of-map m d a = (case m a of Some b  $\Rightarrow$  b | None  $\Rightarrow$  d)

```

```

end

```

```

theory Orthogonality

```

```

imports

```

```

  Critical-Pairs

```

```

begin

```

This theory contains the result, that weak orthogonality implies confluence

We prove the diamond property of *par-rstep* for weakly orthogonal systems.

```

context

```

```

  fixes ren :: 'v :: infinite renaming2

```

```

begin

```

```

lemma weakly-orthogonal-main: fixes R :: ('f, 'v)trs

```

```

  assumes st1: (s, t1)  $\in$  par-rstep R and st2: (s, t2)  $\in$  par-rstep R and weak-ortho:
  left-linear-trs R  $\wedge$  b l r. (b, l, r)  $\in$  critical-pairs ren R R  $\Longrightarrow$  l = r

```

```

  and wf:  $\bigwedge$  l r. (l, r)  $\in$  R  $\Longrightarrow$  is-Fun l

```

```

  shows  $\exists$  u. (t1, u)  $\in$  par-rstep R  $\wedge$  (t2, u)  $\in$  par-rstep R

```

```

  <proof>

```

```

lemma weakly-orthogonal-par-rstep-CR:

```

```

  assumes weak-ortho: left-linear-trs R  $\wedge$  b l r. (b, l, r)  $\in$  critical-pairs ren R R
   $\Longrightarrow$  l = r

```

```

  and wf:  $\bigwedge$  l r. (l, r)  $\in$  R  $\Longrightarrow$  is-Fun l

```

```

shows CR (par-rstep R)

```

```

  <proof>

```

```

lemma weakly-orthogonal-rstep-CR:

```

```

  assumes weak-ortho: left-linear-trs R  $\wedge$  b l r. (b, l, r)  $\in$  critical-pairs ren R R
   $\Longrightarrow$  l = r

```

```

  and wf:  $\bigwedge$  l r. (l, r)  $\in$  R  $\Longrightarrow$  is-Fun l

```

```

shows CR (rstep R)

```

```

  <proof>

```

```

end

```

```

end

```

```

theory Par-Step-Var-Restricted

```

```

imports

```

```

  Trs

```

```

  Multihole-Context

```

```

  SubList

```

begin

fun *vars-below-hole* :: ('f,'v)term \Rightarrow ('f,'v)mctxt \Rightarrow 'v set **where**
 vars-below-hole *t* *MHole* = *vars-term* *t*
 | *vars-below-hole* *t* (*MVar* *y*) = {}
 | *vars-below-hole* (*Fun* - *ts*) (*MFun* - *Cs*) =
 \bigcup (set (map (λ (*t*, *C*). *vars-below-hole* *t* *C*) (zip *ts* *Cs*)))
 | *vars-below-hole* (*Var* -) (*MFun* -) = *Code.abort* (*STR* "assumption in vars-below-hole
 violated") (λ -. {})

lemma *vars-below-hole-no-hole*: *hole-poss* *C* = {} \implies *vars-below-hole* *t* *C* = {}
 <proof>

lemma *vars-below-hole-mctxt-of-term[simp]*: *vars-below-hole* *t* (*mctxt-of-term* *u*) = {}
 <proof>

lemma *vars-below-hole-vars-term*: *vars-below-hole* *t* *C* \subseteq *vars-term* *t*
 <proof>

lemma *vars-below-hole-subst[simp]*: *vars-below-hole* *t* (*C* · *mc* σ) = *vars-below-hole* *t* *C*
 <proof>

lemma *vars-below-hole-Fun*: **assumes** *length* *ls* = *length* *Cs*
shows *vars-below-hole* (*Fun* *f* *ls*) (*MFun* *f* *Cs*) = \bigcup {*vars-below-hole* (*ls* ! *i*) (*Cs* ! *i*) | *i*. *i* < *length* *Cs*}
 <proof>

lemma *vars-below-hole-term-subst*:
hole-poss *D* \subseteq *poss* *t* \implies *vars-below-hole* (*t* · σ) *D* = \bigcup (*vars-term* ' σ ' *vars-below-hole* *t* *D*)
 <proof>

lemma *vars-below-hole-egf*: **assumes** *t* =_{*f*} (*C*, *ts*)
shows *vars-below-hole* *t* *C* = \bigcup (*vars-term* ' set *ts*)
 <proof>

definition *par-rstep-var-restr* *R* *V* = {(*s*, *t*) | *s* *t* *C* infos.
 (*s*, *t*) \in *par-rstep-mctxt* *R* *C* infos \wedge *vars-below-hole* *t* *C* \cap *V* = {}}

lemma *par-rstep-var-restr-mono*: **assumes** *R* \subseteq *S* *W* \subseteq *V*
shows *par-rstep-var-restr* *R* *V* \subseteq *par-rstep-var-restr* *S* *W*

$\langle \text{proof} \rangle$

lemma *par-rstep-var-restr-refl*[simp]: $(t, t) \in \text{par-rstep-var-restr } R \ V$
 $\langle \text{proof} \rangle$

lemma *merge-par-rstep-var-restr*:
assumes *subst-R*: $\bigwedge x. (\delta x, \gamma x) \in \text{par-rstep } R$
and *st*: $(s, t) \in \text{par-rstep-var-restr } R \ V$
and *subst-eq*: $\bigwedge x. x \notin V \implies \delta x = \gamma x$
shows $(s \cdot \delta, t \cdot \gamma) \in \text{par-rstep } R$
 $\langle \text{proof} \rangle$

the variable restricted parallel rewrite relation is closed under variable renamings, provided that the set of forbidden variables is also renamed (in the inverse way)

lemma *par-rstep-var-restr-subst*:
assumes $(s, t) \in \text{par-rstep-var-restr } R \ (\gamma \text{ ` } V)$
and $\bigwedge x. \sigma x \cdot (\text{Var } o \ \gamma) = \text{Var } x$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-var-restr } R \ V$
 $\langle \text{proof} \rangle$

end

theory *Term-Impl*

imports

First-Order-Terms.Term-More
Certification-Monads.Check-Monad
Deriving.Compare-Order-Instances
Show.Shows-Literal
Preliminaries

begin

derive *compare-order term*

fun *poss-list* :: $(f, 'v) \text{ term} \Rightarrow \text{pos list}$

where

poss-list (Var *x*) = $[\] \mid$
poss-list (Fun *f ss*) = $([\] \# \text{concat } (\text{map } (\lambda (i, ps). \text{map } ((\#) \ i) \ ps) \ (\text{zip } [0 \ ..< \ \text{length } \ ss] \ (\text{map } \text{poss-list } \ ss))))$

lemma *poss-list-sound* [simp]:

set (*poss-list s*) = *poss s*

$\langle \text{proof} \rangle$

declare *poss-list.simps* [simp del]

fun *var-poss-list* :: $(f, 'v) \text{ term} \Rightarrow \text{pos list}$

where

var-poss-list (*Var x*) = [] |
var-poss-list (*Fun f ss*) = (concat (map (λ (*i*, *ps*).
 map ((#) *i*) *ps*) (zip [0 ..< length *ss*] (map *var-poss-list ss*))))

lemma *var-poss-list-sound* [simp]:

set (*var-poss-list s*) = *var-poss s*
 ⟨proof⟩

lemma *length-var-poss-list*: length (*var-poss-list t*) = length (*vars-term-list t*)

⟨proof⟩

lemma *vars-term-list-var-poss-list*:

assumes *i* < length (*vars-term-list t*)
shows Var ((*vars-term-list t*)!i) = t|-(*var-poss-list t*)!i
 ⟨proof⟩

lemma *var-poss-list-map-vars-term*:

shows *var-poss-list* (map-vars-term *f t*) = *var-poss-list t*
 ⟨proof⟩

lemma *distinct-var-poss-list*:

shows distinct (*var-poss-list t*)
 ⟨proof⟩

8 Variables and Variable Positions

Duplicate free version of *vars-term-list* that preserves order of appearance of variables.

abbreviation *vars-distinct* :: ('f, 'v) term ⇒ 'v list **where** *vars-distinct t* ≡ (rev
 ∘ remdups ∘ rev) (*vars-term-list t*)

lemma *single-var*[simp]: *vars-distinct* (*Var x*) = [*x*]

⟨proof⟩

lemma *vars-term-list-vars-distinct*:

assumes *i* < length (*vars-term-list t*)
shows ∃ *j* < length (*vars-distinct t*). (*vars-term-list t*)!i = (*vars-distinct t*)!j
 ⟨proof⟩

context

begin

private fun *in-poss* :: pos ⇒ ('f, 'v) term ⇒ bool

where

in-poss [] - ⟷ True |
in-poss (Cons *i p*) (*Fun f ts*) ⟷ *i* < length *ts* ∧ *in-poss p* (*ts* ! *i*) |
in-poss (Cons *i p*) (*Var -*) ⟷ False

```

lemma poss-code[code-unfold]:
   $p \in \text{poss } t = \text{in-poss } p \ t \ \langle \text{proof} \rangle$ 
end

```

8.1 Useful abstractions

Given that we perform the same operations on terms in order to get a list of the variables and a list of the functions, we define functions that run through the term and perform these actions.

```

context
begin
private fun contains-var-term :: ('f, 'v) term  $\Rightarrow$  bool where
  contains-var-term  $x \ (Var \ y) = (x = y)$ 
| contains-var-term  $x \ (Fun \ - \ ts) = Bex \ (set \ ts) \ (contains-var-term \ x)$ 

```

```

lemma contains-var-term-sound[simp]:
   $contains-var-term \ x \ t \longleftrightarrow x \in vars-term \ t$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma in-vars-term-code[code-unfold]:  $x \in vars-term \ t = contains-var-term \ x \ t$ 
   $\langle \text{proof} \rangle$ 
end

```

```

lemma linear-vars-term-list:
  assumes linear-term  $t$ 
  shows  $length \ (filter \ ((=) \ x) \ (vars-term-list \ t)) \leq 1$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma linear-term-distinct-vars:
  assumes linear-term  $t$ 
  shows distinct  $(vars-term-list \ t)$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma distinct-vars-linear-term:
  assumes distinct  $(vars-term-list \ t)$ 
  shows linear-term  $t$ 
   $\langle \text{proof} \rangle$ 

```

```

fun supteq-list :: ('f, 'v) term  $\Rightarrow$  ('f, 'v) term list
where
  supteq-list  $(Var \ x) = [Var \ x] \mid$ 
  supteq-list  $(Fun \ f \ ts) = Fun \ f \ ts \ \# \ concat \ (map \ supteq-list \ ts)$ 

```

```

fun supt-list :: ('f, 'v) term  $\Rightarrow$  ('f, 'v) term list
where
  supt-list  $(Var \ x) = [] \mid$ 

```

$\text{supt-list } (\text{Fun } f \text{ ts}) = \text{concat } (\text{map } \text{supteq-list } \text{ts})$

lemma *supteq-list [simp]*:
 $\text{set } (\text{supteq-list } t) = \{s. t \supseteq s\}$
 $\langle \text{proof} \rangle$

lemma *supt-list-sound [simp]*:
 $\text{set } (\text{supt-list } t) = \{s. t \supset s\}$
 $\langle \text{proof} \rangle$

fun *supt-impl* :: (*f*, *v*) *term* \Rightarrow (*f*, *v*) *term* \Rightarrow *bool*
where
 $\text{supt-impl } (\text{Var } x) \text{ } t \longleftrightarrow \text{False} \mid$
 $\text{supt-impl } (\text{Fun } f \text{ ss}) \text{ } t \longleftrightarrow t \in \text{set } \text{ss} \vee \text{Bex } (\text{set } \text{ss}) (\lambda s. \text{supt-impl } s \text{ } t)$

lemma *supt-impl [code-unfold]*:
 $s \supset t \longleftrightarrow \text{supt-impl } s \text{ } t$
 $\langle \text{proof} \rangle$

lemma *supteq-impl [code-unfold]*: $s \supseteq t \longleftrightarrow s = t \vee \text{supt-impl } s \text{ } t$
 $\langle \text{proof} \rangle$

fun
 $\text{linear-term-impl} :: 'v \text{ set} \Rightarrow ('f, 'v) \text{ term} \Rightarrow ('v \text{ set}) \text{ option}$
where
 $\text{linear-term-impl } \text{xs } (\text{Var } x) = (\text{if } x \in \text{xs} \text{ then } \text{None} \text{ else } \text{Some } (\text{insert } x \text{ xs})) \mid$
 $\text{linear-term-impl } \text{xs } (\text{Fun } - []) = \text{Some } \text{xs} \mid$
 $\text{linear-term-impl } \text{xs } (\text{Fun } f \text{ (} t \# \text{ts)}) = (\text{case } \text{linear-term-impl } \text{xs } t \text{ of}$
 $\text{None} \Rightarrow \text{None}$
 $\mid \text{Some } \text{ys} \Rightarrow \text{linear-term-impl } \text{ys } (\text{Fun } f \text{ ts}))$

lemma *linear-term-code [code]*: $\text{linear-term } t = (\text{linear-term-impl } \{\} \text{ } t \neq \text{None})$
 $\langle \text{proof} \rangle$

definition *check-no-var* :: (*f*::*showl*, *v*::*showl*) *term* \Rightarrow *showsl check* **where**
 $\text{check-no-var } t \equiv \text{check } (\text{is-Fun } t) (\text{showsl } (\text{STR } \text{"variable found"} \sqsupset))$

lemma *check-no-var-sound [simp]*:
 $\text{isOK } (\text{check-no-var } t) \longleftrightarrow \text{is-Fun } t$
 $\langle \text{proof} \rangle$

definition
 $\text{check-supt} :: ('f::\text{showl}, 'v::\text{showl}) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{showsl check}$
where
 $\text{check-supt } s \text{ } t \equiv \text{check } (s \supset t) (\text{showsl } t \circ \text{showsl } (\text{STR } \text{"is not a proper subterm of "}) \circ \text{showsl } s)$

definition

$check\text{-}supteq :: ('f::showl, 'v::showl) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow showsl \text{ check}$
where
 $check\text{-}supteq \ s \ t \equiv check \ (s \supseteq t) \ (showsl \ t \circ showsl \ (STR \ " \text{ is not a subterm of }"$
 $\ "') \circ showsl \ s)$

lemma *isOK-check-supt* [simp]:
 $isOK \ (check\text{-}supt \ s \ t) \longleftrightarrow s \triangleright t$
 $\langle proof \rangle$

lemma *isOK-check-supteq* [simp]:
 $isOK \ (check\text{-}supteq \ s \ t) \longleftrightarrow s \supseteq t$
 $\langle proof \rangle$

8.2 Additional Functions on Terms

fun *with-arity* :: $('f, 'v) \text{ term} \Rightarrow ('f \times nat, 'v) \text{ term}$ **where**
 $with\text{-}arity \ (Var \ x) = Var \ x$
 $| \ with\text{-}arity \ (Fun \ f \ ts) = Fun \ (f, length \ ts) \ (map \ with\text{-}arity \ ts)$

fun *add-vars-term* :: $('f, 'v) \text{ term} \Rightarrow 'v \text{ list} \Rightarrow 'v \text{ list}$
where
 $add\text{-}vars\text{-}term \ (Var \ x) \ xs = x \# \ xs \ |$
 $add\text{-}vars\text{-}term \ (Fun \ f \ ts) \ xs = foldr \ add\text{-}vars\text{-}term \ ts \ xs$

fun *add-funs-term* :: $('f, 'v) \text{ term} \Rightarrow 'f \text{ list} \Rightarrow 'f \text{ list}$
where
 $add\text{-}funs\text{-}term \ (Var \ -) \ fs = fs \ |$
 $add\text{-}funs\text{-}term \ (Fun \ f \ ts) \ fs = f \# \ foldr \ add\text{-}funs\text{-}term \ ts \ fs$

fun *add-funas-term* :: $('f, 'v) \text{ term} \Rightarrow ('f \times nat) \text{ list} \Rightarrow ('f \times nat) \text{ list}$
where
 $add\text{-}funas\text{-}term \ (Var \ -) \ fs = fs \ |$
 $add\text{-}funas\text{-}term \ (Fun \ f \ ts) \ fs = (f, length \ ts) \# \ foldr \ add\text{-}funas\text{-}term \ ts \ fs$

definition *add-funas-args-term* :: $('f, 'v) \text{ term} \Rightarrow ('f \times nat) \text{ list} \Rightarrow ('f \times nat) \text{ list}$
where
 $add\text{-}funas\text{-}args\text{-}term \ t \ fs = foldr \ add\text{-}funas\text{-}term \ (args \ t) \ fs$

lemma *add-vars-term-vars-term-list-conv* [simp]:
 $add\text{-}vars\text{-}term \ t \ xs = vars\text{-}term\text{-}list \ t \ @ \ xs$
 $\langle proof \rangle$

lemma *add-funs-term-funs-term-list-conv* [simp]:
 $add\text{-}funs\text{-}term \ t \ fs = funs\text{-}term\text{-}list \ t \ @ \ fs$
 $\langle proof \rangle$

lemma *add-funas-term-funas-term-list-conv* [simp]:
 $add\text{-}funas\text{-}term \ t \ fs = funas\text{-}term\text{-}list \ t \ @ \ fs$
 $\langle proof \rangle$

lemma *add-vars-term-vars-term-list-abs-conv* [simp]:

add-vars-term = ($@$) \circ *vars-term-list*

\langle proof \rangle

lemma *add-funs-term-funs-term-list-abs-conv* [simp]:

add-funs-term = ($@$) \circ *funs-term-list*

\langle proof \rangle

lemma *add-funas-term-funas-term-list-abs-conv* [simp]:

add-funas-term = ($@$) \circ *funas-term-list*

\langle proof \rangle

lemma *add-funas-args-term-funas-args-term-list-conv* [simp]:

add-funas-args-term *t* *fs* = *funas-args-term-list* *t* $@$ *fs*

\langle proof \rangle

fun *insert-vars-term* :: (*f*, *v*) *term* \Rightarrow *v* *list* \Rightarrow *v* *list*

where

insert-vars-term (*Var* *x*) *xs* = *List.insert* *x* *xs* |

insert-vars-term (*Fun* *f* *ts*) *xs* = *foldr insert-vars-term* *ts* *xs*

fun *insert-funs-term* :: (*f*, *v*) *term* \Rightarrow *f* *list* \Rightarrow *f* *list*

where

insert-funs-term (*Var* *x*) *fs* = *fs* |

insert-funs-term (*Fun* *f* *ts*) *fs* = *List.insert* *f* (*foldr insert-funs-term* *ts* *fs*)

fun *insert-funas-term* :: (*f*, *v*) *term* \Rightarrow (*f* \times *nat*) *list* \Rightarrow (*f* \times *nat*) *list*

where

insert-funas-term (*Var* *x*) *fs* = *fs* |

insert-funas-term (*Fun* *f* *ts*) *fs* = *List.insert* (*f*, *length* *ts*) (*foldr insert-funas-term* *ts* *fs*)

definition *insert-funas-args-term* :: (*f*, *v*) *term* \Rightarrow (*f* \times *nat*) *list* \Rightarrow (*f* \times *nat*) *list*

where

insert-funas-args-term *t* *fs* = *foldr insert-funas-term* (*args* *t*) *fs*

lemma *set-insert-vars-term-vars-term* [simp]:

set (*insert-vars-term* *t* *xs*) = *vars-term* *t* \cup *set* *xs*

\langle proof \rangle

lemma *set-insert-funs-term-funs-term* [simp]:

set (*insert-funs-term* *t* *fs*) = *funs-term* *t* \cup *set* *fs*

\langle proof \rangle

lemma *set-insert-funas-term-funas-term* [simp]:

set (*insert-funas-term* *t* *fs*) = *funas-term* *t* \cup *set* *fs*

\langle proof \rangle

lemma *set-insert-funas-args-term* [simp]:
 $set (insert-funas-args-term t fs) = \bigcup (funas-term \text{ ` } set (args t)) \cup set fs$
 <proof>

Implementations of corresponding set-based functions.

abbreviation *vars-term-impl* $t \equiv insert-vars-term t []$
abbreviation *funas-term-impl* $t \equiv insert-funas-term t []$
abbreviation *funas-term-impl* $t \equiv insert-funas-term t []$

lemma [code]:
 $vars-term-list t = add-vars-term t []$
 $funas-term-list t = add-funas-term t []$
 <proof>

lemma *with-arity-term-fold* [code]:
 $with-arity = Term-More.fold Var (\lambda f ts. Fun (f, length ts) ts)$
 <proof>

fun *flatten-term-enum* :: $(f list, 'v) term \Rightarrow (f, 'v) term list$
where
 $flatten-term-enum (Var x) = [Var x] |$
 $flatten-term-enum (Fun fs ts) =$
 (let
 $lts = map flatten-term-enum ts;$
 $ss = concat-lists lts$
 in $concat (map (\lambda f. map (Fun f) ss) fs))$

lemma *flatten-term-enum*:
 $set (flatten-term-enum t) = \{u. instance-term u (map-funs-term set t)\}$
 <proof>

definition *mk-subst-domain* :: $(f, 'v) substL \Rightarrow ('v \times (f, 'v) term) list$ **where**
 $mk-subst-domain \sigma \equiv$
 $let \tau = mk-subst Var \sigma in$
 $(filter (\lambda(x, t). Var x \neq t) (map (\lambda x. (x, \tau x)) (remdups (map fst \sigma))))$

lemma *mk-subst-domain*:
 $set (mk-subst-domain \sigma) = (\lambda x. (x, mk-subst Var \sigma x)) \text{ ` } subst-domain (mk-subst Var \sigma)$
 (is ?I = ?R)
 <proof>

lemma *finite-mk-subst*: $finite (subst-domain (mk-subst Var \sigma))$
 <proof>

definition *subst-eq* :: $(f, 'v) substL \Rightarrow (f, 'v) substL \Rightarrow bool$ **where**

$\text{subst-eq } \sigma \tau = (\text{let } \sigma' = \text{mk-subst-domain } \sigma; \tau' = \text{mk-subst-domain } \tau \text{ in set } \sigma' = \text{set } \tau')$

lemma *subst-eq* [simp]:

$\text{subst-eq } \sigma \tau = (\text{mk-subst Var } \sigma = \text{mk-subst Var } \tau)$
 $\langle \text{proof} \rangle$

definition *range-vars-impl* :: $(f, v) \text{ substL} \Rightarrow v \text{ list}$
where

$\text{range-vars-impl } \sigma =$
 $(\text{let } \sigma' = \text{mk-subst-domain } \sigma \text{ in}$
 $\text{concat } (\text{map } (\text{vars-term-list } o \text{ snd}) \sigma'))$

definition *vars-subst-impl* :: $(f, v) \text{ substL} \Rightarrow v \text{ list}$
where

$\text{vars-subst-impl } \sigma =$
 $(\text{let } \sigma' = \text{mk-subst-domain } \sigma \text{ in}$
 $\text{map fst } \sigma' @ \text{concat } (\text{map } (\text{vars-term-list } o \text{ snd}) \sigma'))$

lemma *vars-subst-impl* [simp]:

$\text{set } (\text{vars-subst-impl } \sigma) = \text{vars-subst } (\text{mk-subst Var } \sigma)$
 $\langle \text{proof} \rangle$

lemma *range-vars-impl* [simp]:

$\text{set } (\text{range-vars-impl } \sigma) = \text{range-vars } (\text{mk-subst Var } \sigma)$
 $\langle \text{proof} \rangle$

lemma *mk-subst-one* [simp]: $\text{mk-subst Var } [(x, t)] = \text{subst } x \ t$
 $\langle \text{proof} \rangle$

lemma *fst-image* [simp]: $\text{fst } '(\lambda x. (x, g \ x)) \ 'a = a \ \langle \text{proof} \rangle$

definition

$\text{subst-compose-impl} :: (f, v) \text{ substL} \Rightarrow (f, v) \text{ substL} \Rightarrow (f, v) \text{ substL}$

where

$\text{subst-compose-impl } \sigma \tau \equiv$
 let
 $\sigma' = \text{mk-subst-domain } \sigma;$
 $\tau' = \text{mk-subst-domain } \tau;$
 $d\sigma = \text{map fst } \sigma'$
 $\text{in map } (\lambda (x, t). (x, t \cdot \text{mk-subst Var } \tau')) \sigma' @ \text{filter } (\lambda (x, t). x \notin \text{set } d\sigma) \tau'$

lemma *mk-subst-mk-subst-domain* [simp]:

$\text{mk-subst Var } (\text{mk-subst-domain } \sigma) = \text{mk-subst Var } \sigma$
 $\langle \text{proof} \rangle$

lemma *subst-compose-impl* [simp]:

$\text{mk-subst Var } (\text{subst-compose-impl } \sigma \tau) = \text{mk-subst Var } \sigma \circ_s \text{mk-subst Var } \tau$ (**is**
 $?l = ?r$)

$\langle \text{proof} \rangle$

fun *subst-power-impl* :: (*f*, *v*) *substL* \Rightarrow *nat* \Rightarrow (*f*, *v*) *substL* **where**
subst-power-impl σ 0 = []
| *subst-power-impl* σ (Suc *n*) = *subst-compose-impl* σ (*subst-power-impl* σ *n*)

lemma *subst-power-impl [simp]*:
mk-subst *Var* (*subst-power-impl* σ *n*) = (*mk-subst* *Var* σ) \sim_n
 $\langle \text{proof} \rangle$

definition *commutes-impl* :: (*f*, *v*) *substL* \Rightarrow (*f*, *v*) *substL* \Rightarrow *bool* **where**
commutes-impl σ μ \equiv *subst-eq* (*subst-compose-impl* σ μ) (*subst-compose-impl* μ σ)

lemma *commutes-impl [simp]*:
commutes-impl σ μ = ((*mk-subst* *Var* σ \circ_s *mk-subst* *Var* μ) = (*mk-subst* *Var* μ \circ_s *mk-subst* *Var* σ))
 $\langle \text{proof} \rangle$

definition
subst-compose'-impl :: (*f*, *v*) *substL* \Rightarrow (*f*, *v*) *subst* \Rightarrow (*f*, *v*) *substL*
where
subst-compose'-impl σ ϱ \equiv *map* (λ (*x*, *s*). (*x*, *s* \cdot ϱ)) (*mk-subst-domain* σ)

lemma *subst-compose'-impl [simp]*:
mk-subst *Var* (*subst-compose'-impl* σ ϱ) = *subst-compose'* (*mk-subst* *Var* σ) ϱ (**is** $?l = ?r$)
 $\langle \text{proof} \rangle$

definition
subst-replace-impl :: (*f*, *v*) *substL* \Rightarrow *v* \Rightarrow (*f*, *v*) *term* \Rightarrow (*f*, *v*) *substL*
where
subst-replace-impl σ *x* *t* \equiv (*x*, *t*) # *filter* (λ (*y*, *t*). *y* \neq *x*) σ

lemma *subst-replace-impl [simp]*:
mk-subst *Var* (*subst-replace-impl* σ *x* *t*) = (λ *y*. if *x* = *y* then *t* else *mk-subst* *Var* σ *y*) (**is** $?l = ?r$)
 $\langle \text{proof} \rangle$

lemma *mk-subst-domain-distinct*: *distinct* (*map* *fst* (*mk-subst-domain* σ))
 $\langle \text{proof} \rangle$

definition *is-renaming-impl* :: (*f*, *v*) *substL* \Rightarrow *bool* **where**
is-renaming-impl σ \equiv
let $\sigma' = \text{map } \text{snd } (\text{mk-subst-domain } \sigma)$ *in*
 $(\forall t \in \text{set } \sigma'. \text{is-Var } t) \wedge \text{distinct } \sigma'$

lemma *is-renaming-impl [simp]*:

is-renaming-impl $\sigma = \text{is-renaming } (\text{mk-subst Var } \sigma) \text{ (is ?l = ?r)}$
 $\langle \text{proof} \rangle$

definition *is-inverse-renaming-impl* :: $(f, 'v) \text{ substL} \Rightarrow (f, 'v) \text{ substL}$ **where**
is-inverse-renaming-impl $\sigma \equiv$
 let $\sigma' = \text{mk-subst-domain } \sigma$ in
 map $(\lambda (x, y). (\text{the-Var } y, \text{Var } x)) \sigma'$

lemma *is-inverse-renaming-impl* [simp]:
fixes $\sigma :: (f, 'v) \text{ substL}$
assumes *var*: *is-renaming* $(\text{mk-subst Var } \sigma)$
shows $\text{mk-subst Var } (\text{is-inverse-renaming-impl } \sigma) = \text{is-inverse-renaming } (\text{mk-subst Var } \sigma) \text{ (is ?l = ?r)}$
 $\langle \text{proof} \rangle$

definition
mk-subst-case :: $'v \text{ list} \Rightarrow (f, 'v) \text{ subst} \Rightarrow (f, 'v) \text{ substL} \Rightarrow (f, 'v) \text{ substL}$
where
mk-subst-case $xs \sigma \tau = \text{subst-compose-impl } (\text{map } (\lambda x. (x, \sigma x)) xs) \tau$

lemma *mk-subst-case* [simp]:
 $\text{mk-subst Var } (\text{mk-subst-case } xs \sigma \tau) =$
 $(\lambda x. \text{if } x \in \text{set } xs \text{ then } \sigma x \cdot \text{mk-subst Var } \tau \text{ else } \text{mk-subst Var } \tau x)$
 $\langle \text{proof} \rangle$

definition *check-linear-term* :: $(f :: \text{showl}, 'v :: \text{showl}) \text{ term} \Rightarrow \text{showsl check}$
where
check-linear-term $s = \text{check } (\text{linear-term } s)$
 $(\text{showsl } (\text{STR "the term "}) \circ \text{showsl } s \circ \text{showsl } (\text{STR " is not linear" } \boxed{\longleftrightarrow}))$

lemma *check-linear-term* [simp]:
isOK $(\text{check-linear-term } s) = \text{linear-term } s$
 $\langle \text{proof} \rangle$

definition *check-ground-term* :: $(f :: \text{showl}, 'v :: \text{showl}) \text{ term} \Rightarrow \text{showsl check}$
where
check-ground-term $s = \text{check } (\text{ground } s)$
 $(\text{showsl } (\text{STR "the term "}) \circ \text{showsl } s \circ \text{showsl } (\text{STR " is not a ground term" } \boxed{\longleftrightarrow}))$

lemma *check-ground-term* [simp]:
isOK $(\text{check-ground-term } s) \longleftrightarrow \text{ground } s$
 $\langle \text{proof} \rangle$

type-synonym *f sig-list* = $(f \times \text{nat})\text{list}$

fun *check-funas-term* :: $f :: \text{showl sig} \Rightarrow (f, 'v :: \text{showl}) \text{ term} \Rightarrow \text{showsl check}$ **where**
check-funas-term $F (\text{Fun } f \text{ ts}) = \text{do } \{$

```

      check ((f, length ts) ∈ F) (showsl (Fun f ts)
      o showsl-lit (STR "problem: root of subterm  ") o showsl f o showsl-lit (STR
" not in signature" <math>\Leftarrow</math> '));
      check-allm (check-funas-term F) ts
    }
  | check-funas-term F (Var -) = return ()

```

lemma *check-funas-term[simp]: isOK(check-funas-term F t) = (funas-term t ⊆ F)*
 ⟨proof⟩

end

theory *Trs-Impl*
imports
Trs
First-Order-Rewriting.Term-Impl
First-Order-Terms.Matching
First-Order-Rewriting.Abstract-Rewriting-Impl
Option-Util
Transitive-Closure.RBT-Map-Set-Extension
begin

type-synonym (*f*, *v*) *rules* = (*f*, *v*) *rule list*

context fixes *R* :: (*f*, *v*) *rules*
begin

definition *rrewrite* :: (*f*, *v*) *term* ⇒ (*f*, *v*) *term list*
where

```

  rrewrite s = List.maps (λ (l, r) . case match s l of
    None ⇒ []
  | Some σ ⇒ [r · σ]) R

```

lemma *rrewrite-sound*: *t* ∈ *set* (*rrewrite* *s*) ⇒ (*s*, *t*) ∈ *rrstep* (*set* *R*)
 ⟨proof⟩

lemma *rrewrite-complete*: **assumes** (*s*, *t*) ∈ *rrstep* (*set* *R*)
shows ∃ *u*. *u* ∈ *set* (*rrewrite* *s*)
 ⟨proof⟩

lemma *rrewrite*: **assumes** $\bigwedge l r. (l, r) \in \text{set } R \implies \text{vars-term } l \supseteq \text{vars-term } r$
shows *set* (*rrewrite* *s*) = {*t*. (*s*, *t*) ∈ *rrstep* (*set* *R*)}
 ⟨proof⟩

fun *rewrite* :: (*f*, *v*) *term* ⇒ (*f*, *v*) *term list* **where**
rewrite *s* = (*rrewrite* *s* @ (case *s* of Var - ⇒ [] | Fun *f* *ss* ⇒
 concat (map (λ (*i*, *si*). map (λ *ti*. Fun *f* (*ss*[*i* := *ti*])) (*rewrite* *si*))
 (zip [0..*length* *ss*] *ss*))))

declare *rewrite.simps*[*simp del*]

lemma *rewrite-sound*: $t \in \text{set } (\text{rewrite } s) \implies (s, t) \in \text{rstep } (\text{set } R)$
 $\langle \text{proof} \rangle$

lemma *rewrite*: **assumes** $\bigwedge l r. (l, r) \in \text{set } R \implies \text{vars-term } l \supseteq \text{vars-term } r$
shows $\text{set } (\text{rewrite } s) = \{t. (s, t) \in \text{rstep } (\text{set } R)\}$
 $\langle \text{proof} \rangle$

lemma *rewrite-complete*: **assumes** $(s, t) \in \text{rstep } (\text{set } R)$
shows $\exists w. w \in \text{set } (\text{rewrite } s)$
 $\langle \text{proof} \rangle$
end

lemma *rrewrite-mono*: $\text{set } R \subseteq \text{set } S \implies \text{set } (\text{rrewrite } R \ s) \subseteq \text{set } (\text{rrewrite } S \ s)$
 $\langle \text{proof} \rangle$

lemma *Union-image-mono*: $(\bigwedge x. x \in A \implies f \ x \subseteq g \ x) \implies \bigcup (f \ ` \ A) \subseteq \bigcup (g \ ` \ A)$
 $\langle \text{proof} \rangle$

lemma *rewrite-mono*: **assumes** $\text{set } R \subseteq \text{set } S$
shows $\text{set } (\text{rewrite } R \ s) \subseteq \text{set } (\text{rewrite } S \ s)$
 $\langle \text{proof} \rangle$

definition *first-rewrite* :: $(f, v) \text{rules} \Rightarrow (f, v) \text{term} \Rightarrow (f, v) \text{term option}$
where *first-rewrite* $R \ s \equiv \text{case } \text{rewrite } R \ s \text{ of } \text{Nil} \Rightarrow \text{None} \mid \text{Cons } t \ - \Rightarrow \text{Some } t$

definition *is-root-step* :: $(f, v) \text{trs} \Rightarrow (f, v) \text{term} \Rightarrow (f, v) \text{term} \Rightarrow \text{bool}$
where
is-root-step $R \ s \ t = (\exists (l, r) \in R. \text{case match-list Var } [(l, s), (r, t)] \text{ of}$
 $\text{None} \Rightarrow \text{False}$
 $\mid \text{Some } - \Rightarrow \text{True})$

lemma *rrstep-code*[*code-unfold*]: $(s, t) \in \text{rrstep } R \longleftrightarrow \text{is-root-step } R \ s \ t$
 $\langle \text{proof} \rangle$

lemma *is-root-step*: $\text{is-root-step } R \ s \ t \implies (s, t) \in \text{rrstep } R$
 $\langle \text{proof} \rangle$

fun *is-rstep* :: $(f, v) \text{trs} \Rightarrow (f, v) \text{term} \Rightarrow (f, v) \text{term} \Rightarrow \text{bool}$ **where**
is-rstep $R \ (\text{Fun } f \ ts) \ (\text{Fun } g \ ss) =$
 $f = g \wedge \text{length } ts = \text{length } ss \wedge (\exists i \in \text{set } [0..<\text{length } ss].$
 $ss = ts[i := ss ! i] \wedge \text{is-rstep } R \ (ts ! i) \ (ss ! i))$

$\vee (Fun\ f\ ts, Fun\ g\ ss) \in rstep\ R$
 $| is-rstep\ R\ s\ t = ((s,t) \in rstep\ R)$

lemma *is-rstep-sound*: $is-rstep\ R\ s\ t \implies (s,t) \in rstep\ R$
 $\langle proof \rangle$

lemma *is-rstep-complete*: **assumes** $(s,t) \in rstep\ R$
shows $is-rstep\ R\ s\ t$
 $\langle proof \rangle$

lemma *is-rstep[simp]*: $is-rstep\ R\ s\ t \longleftrightarrow (s,t) \in rstep\ R$
 $\langle proof \rangle$

lemma *in-rstep-code[code-unfold]*:
 $st \in rstep\ R \longleftrightarrow (case\ st\ of\ (s,t) \Rightarrow is-rstep\ R\ s\ t)$
 $\langle proof \rangle$

definition *compute-rstep-NF* :: $(f,v)rules \Rightarrow (f,v)term \Rightarrow (f,v)term\ option$
where $compute-rstep-NF\ R\ s \equiv compute-NF\ (first-rewrite\ R)\ s$

lemma *compute-rstep-NF-sound*:
assumes $res: compute-rstep-NF\ R\ s = Some\ t$
shows $(s, t) \in (rstep\ (set\ R))^* \langle proof \rangle$

lemma *compute-rstep-NF-complete*: **assumes** $res: compute-rstep-NF\ R\ s = Some\ t$
shows $t \in NF\ (rstep\ (set\ R)) \langle proof \rangle$

lemma *compute-rstep-NF-SN*: **assumes** $SN: SN\ (rstep\ (set\ R))$
shows $\exists\ t. compute-rstep-NF\ R\ s = Some\ t$
 $\langle proof \rangle$

definition

check-join-NF ::
 $(f :: showl, v :: showl)\ rules \Rightarrow$
 $(f, v)\ term \Rightarrow (f, v)\ term \Rightarrow showsl\ check$

where

$check-join-NF\ R\ s\ t \equiv case\ (compute-rstep-NF\ R\ s, compute-rstep-NF\ R\ t)\ of$
 $(Some\ s', Some\ t') \Rightarrow$
 $check\ (s' = t')\ ($
 $showsl\ (STR\ "the\ normal\ form\ ") \circ showsl\ s' \circ showsl\ (STR\ "of\ ") \circ showsl$
 s
 $\circ showsl\ (STR\ "differs\ from\ \boxed{\leftarrow}\ the\ normal\ form\ ") \circ showsl\ t' \circ showsl\ (STR$
 $"of\ ") \circ showsl\ t)$
 $| - \Rightarrow error\ (showsl\ (STR\ "strange\ error\ in\ normal\ form\ computation"))$

lemma *check-join-NF-sound*:
assumes $ok: isOK\ (check-join-NF\ R\ s\ t)$
shows $(s, t) \in join\ (rstep\ (set\ R))$

$\langle \text{proof} \rangle$

fun *reachable-terms* ::

$(f, v) \text{ rules} \Rightarrow (f, v) \text{ term} \Rightarrow \text{nat} \Rightarrow (f, v) \text{ term list}$

where

$\text{reachable-terms } R \ s \ 0 = [s]$

| $\text{reachable-terms } R \ s \ (\text{Suc } n) = ($

$\text{let } ts = (\text{reachable-terms } R \ s \ n) \text{ in}$

$\text{remdups } (ts @ (\text{concat } (\text{map } (\lambda t. \text{rewrite } R \ t) \ ts)))$

$)$

lemma *reachable-terms-nat*:

assumes $t \in \text{set } (\text{reachable-terms } R \ s \ i)$

shows $\exists j. j \leq i \wedge (s, t) \in (\text{rstep } (\text{set } R))^\sim j$

$\langle \text{proof} \rangle$

lemma *reachable-terms*:

assumes $t \in \text{set } (\text{reachable-terms } R \ s \ i)$

shows $(s, t) \in (\text{rstep } (\text{set } R))^\wedge^*$

$\langle \text{proof} \rangle$

lemma *reachable-terms-one*:

assumes $t \in \text{set } (\text{reachable-terms } R \ s \ (\text{Suc } 0))$

shows $(s, t) \in (\text{rstep } (\text{set } R))^\wedge =$

$\langle \text{proof} \rangle$

function *iterative-join-search-main* ::

$(f, v) \text{ rules} \Rightarrow (f, v) \text{ term} \Rightarrow (f, v) \text{ term} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$

where

$\text{iterative-join-search-main } R \ s \ t \ i \ n = (\text{if } i \leq n \text{ then}$

$((\text{list-inter } (\text{reachable-terms } R \ s \ i) (\text{reachable-terms } R \ t \ i)) \neq []) \vee (\text{iterative-join-search-main } R \ s \ t \ (\text{Suc } i) \ n)) \text{ else False})$

$\langle \text{proof} \rangle$

termination $\langle \text{proof} \rangle$

lemma *iterative-join-search-main*:

$\text{iterative-join-search-main } R \ s \ t \ i \ n \Longrightarrow (s, t) \in \text{join } (\text{rstep } (\text{set } R))$

$\langle \text{proof} \rangle$

definition *iterative-join-search* **where**

$\text{iterative-join-search } R \ s \ t \ n \equiv \text{iterative-join-search-main } R \ s \ t \ 0 \ n$

lemma *iterative-join-search*: $\text{iterative-join-search } R \ s \ t \ n \Longrightarrow (s, t) \in \text{join } (\text{rstep } (\text{set } R))$

$\langle \text{proof} \rangle$

definition

check-join-BFS-limit ::
 nat \Rightarrow ('f :: showl, 'v :: showl) rules \Rightarrow
 ('f, 'v) term \Rightarrow ('f, 'v) term \Rightarrow showsl check
where
check-join-BFS-limit n R s t \equiv check (iterative-join-search R s t n)
 (showsl (STR "could not find a joining sequence of length at most ") \circ
 showsl n \circ showsl (STR " for the terms ") \circ showsl s \circ
 showsl (STR " and ") \circ showsl t \circ showsl-nl)

lemma *check-join-BFS-limit-sound*:
 assumes ok: isOK (check-join-BFS-limit n R s t)
 shows (s, t) \in join (rstep (set R))
 <proof>

definition *map-funs-rules* :: ('f \Rightarrow 'g) \Rightarrow ('f, 'v) rules \Rightarrow ('g, 'v) rules **where**
map-funs-rules fg R = map (map-funs-rule fg) R

lemma *map-funs-rules-sound*[simp]:
 set (map-funs-rules fg R) = map-funs-trs fg (set R)
 <proof>

8.3 Output

fun *showsl-rule'* :: ('f \Rightarrow showsl) \Rightarrow ('v \Rightarrow showsl) \Rightarrow String.literal \Rightarrow ('f, 'v) rule
 \Rightarrow showsl

where
showsl-rule' fun var arr (l, r) =
 showsl-term' fun var l \circ showsl arr \circ showsl-term' fun var r

definition *showsl-rule* \equiv *showsl-rule'* showsl showsl (STR " -> ")

definition *showsl-weak-rule* \equiv *showsl-rule'* showsl showsl (STR " ->= ")

definition

showsl-rules' :: ('f \Rightarrow showsl) \Rightarrow ('v \Rightarrow showsl) \Rightarrow String.literal \Rightarrow ('f, 'v) rules
 \Rightarrow showsl

where

showsl-rules' fun var arr trs =
 showsl-list-gen (*showsl-rule'* fun var arr) (STR "'") (STR "'") (STR "' \leftarrow '")
 (STR "'") trs \circ showsl-nl

definition *showsl-rules* \equiv *showsl-rules'* showsl showsl (STR " -> ")

definition *showsl-weak-rules* \equiv *showsl-rules'* showsl showsl (STR " ->= ")

definition

showsl-trs' :: ('f \Rightarrow showsl) \Rightarrow ('v \Rightarrow showsl) \Rightarrow String.literal \Rightarrow String.literal \Rightarrow
 ('f, 'v) rules \Rightarrow showsl

where

showsl-trs' fun var name arr R = showsl name \circ showsl (STR "' \leftarrow \leftarrow '") \circ
showsl-rules' fun var arr R

definition $\text{showsl-trs} \equiv \text{showsl-trs}' \text{ showsl showsl } (STR \text{ "rewrite system:"}) (STR \text{ " } \rightarrow \text{ "})$

definition $\text{add-vars-rule} :: ('f, 'v) \text{ rule} \Rightarrow 'v \text{ list} \Rightarrow 'v \text{ list}$

where

$\text{add-vars-rule } r \text{ xs} = \text{add-vars-term } (\text{fst } r) (\text{add-vars-term } (\text{snd } r) \text{ xs})$

definition $\text{add-funs-rule} :: ('f, 'v) \text{ rule} \Rightarrow 'f \text{ list} \Rightarrow 'f \text{ list}$

where

$\text{add-funs-rule } r \text{ fs} = \text{add-funs-term } (\text{fst } r) (\text{add-funs-term } (\text{snd } r) \text{ fs})$

definition $\text{add-funas-rule} :: ('f, 'v) \text{ rule} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{add-funas-rule } r \text{ fs} = \text{add-funas-term } (\text{fst } r) (\text{add-funas-term } (\text{snd } r) \text{ fs})$

definition $\text{add-roots-rule} :: ('f, 'v) \text{ rule} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{add-roots-rule } r \text{ fs} = \text{root-list } (\text{fst } r) @ \text{root-list } (\text{snd } r) @ \text{fs}$

definition $\text{add-funas-args-rule} :: ('f, 'v) \text{ rule} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{add-funas-args-rule } r \text{ fs} = \text{add-funas-args-term } (\text{fst } r) (\text{add-funas-args-term } (\text{snd } r) \text{ fs})$

lemma $\text{add-vars-rule-vars-rule-list-conv } [\text{simp}]$:

$\text{add-vars-rule } r \text{ xs} = \text{vars-rule-list } r @ \text{xs}$

$\langle \text{proof} \rangle$

lemma $\text{add-funs-rule-funs-rule-list-conv } [\text{simp}]$:

$\text{add-funs-rule } r \text{ fs} = \text{funs-rule-list } r @ \text{fs}$

$\langle \text{proof} \rangle$

lemma $\text{add-funas-rule-funas-rule-list-conv } [\text{simp}]$:

$\text{add-funas-rule } r \text{ fs} = \text{funas-rule-list } r @ \text{fs}$

$\langle \text{proof} \rangle$

lemma $\text{add-roots-rule-roots-rule-list-conv } [\text{simp}]$:

$\text{add-roots-rule } r \text{ fs} = \text{roots-rule-list } r @ \text{fs}$

$\langle \text{proof} \rangle$

lemma $\text{add-funas-args-rule-funas-args-rule-list-conv } [\text{simp}]$:

$\text{add-funas-args-rule } r \text{ fs} = \text{funas-args-rule-list } r @ \text{fs}$

$\langle \text{proof} \rangle$

definition $\text{insert-vars-rule} :: ('f, 'v) \text{ rule} \Rightarrow 'v \text{ list} \Rightarrow 'v \text{ list}$

where

$\text{insert-vars-rule } r \text{ xs} = \text{insert-vars-term } (\text{fst } r) (\text{insert-vars-term } (\text{snd } r) \text{ xs})$

definition $\text{insert-funs-rule} :: ('f, 'v) \text{rule} \Rightarrow 'f \text{list} \Rightarrow 'f \text{list}$

where

$\text{insert-funs-rule } r \text{ fs} = \text{insert-funs-term } (\text{fst } r) (\text{insert-funs-term } (\text{snd } r) \text{ fs})$

definition $\text{insert-funas-rule} :: ('f, 'v) \text{rule} \Rightarrow ('f \times \text{nat}) \text{list} \Rightarrow ('f \times \text{nat}) \text{list}$

where

$\text{insert-funas-rule } r \text{ fs} = \text{insert-funas-term } (\text{fst } r) (\text{insert-funas-term } (\text{snd } r) \text{ fs})$

definition $\text{insert-roots-rule} :: ('f, 'v) \text{rule} \Rightarrow ('f \times \text{nat}) \text{list} \Rightarrow ('f \times \text{nat}) \text{list}$

where

$\text{insert-roots-rule } r \text{ fs} =$

$\text{foldr List.insert } (\text{option-to-list } (\text{root } (\text{fst } r)) @ \text{option-to-list } (\text{root } (\text{snd } r))) \text{ fs}$

definition $\text{insert-funas-args-rule} :: ('f, 'v) \text{rule} \Rightarrow ('f \times \text{nat}) \text{list} \Rightarrow ('f \times \text{nat}) \text{list}$

where

$\text{insert-funas-args-rule } r \text{ fs} = \text{insert-funas-args-term } (\text{fst } r) (\text{insert-funas-args-term } (\text{snd } r) \text{ fs})$

lemma $\text{set-insert-vars-rule } [\text{simp}]$:

$\text{set } (\text{insert-vars-rule } r \text{ xs}) = \text{vars-term } (\text{fst } r) \cup \text{vars-term } (\text{snd } r) \cup \text{set } xs$

$\langle \text{proof} \rangle$

lemma $\text{set-insert-funs-rule } [\text{simp}]$:

$\text{set } (\text{insert-funs-rule } r \text{ xs}) = \text{funs-term } (\text{fst } r) \cup \text{funs-term } (\text{snd } r) \cup \text{set } xs$

$\langle \text{proof} \rangle$

lemma $\text{set-insert-funas-rule } [\text{simp}]$:

$\text{set } (\text{insert-funas-rule } r \text{ xs}) = \text{funas-term } (\text{fst } r) \cup \text{funas-term } (\text{snd } r) \cup \text{set } xs$

$\langle \text{proof} \rangle$

lemma $\text{set-insert-roots-rule } [\text{simp}]$:

$\text{set } (\text{insert-roots-rule } r \text{ xs}) = \text{root-set } (\text{fst } r) \cup \text{root-set } (\text{snd } r) \cup \text{set } xs$

$\langle \text{proof} \rangle$

lemma $\text{set-insert-funas-args-rule } [\text{simp}]$:

$\text{set } (\text{insert-funas-args-rule } r \text{ xs}) = \text{funas-args-term } (\text{fst } r) \cup \text{funas-args-term } (\text{snd } r) \cup \text{set } xs$

$\langle \text{proof} \rangle$

abbreviation $\text{vars-rule-impl } r \equiv \text{insert-vars-rule } r []$

abbreviation $\text{funs-rule-impl } r \equiv \text{insert-funs-rule } r []$

abbreviation $\text{funas-rule-impl } r \equiv \text{insert-funas-rule } r []$

abbreviation $\text{roots-rule-impl } r \equiv \text{insert-roots-rule } r []$

abbreviation $\text{funas-args-rule-impl } r \equiv \text{insert-funas-args-rule } r []$

lemma $\text{set-vars-rule-impl}$:

$\text{set } (\text{vars-rule-impl } r) = \text{vars-rule } r$

$\langle \text{proof} \rangle$

lemma *[code]*:

vars-rule-list *r* = *add-vars-rule* *r* []
funas-rule-list *r* = *add-funs-rule* *r* []
funas-rule-list *r* = *add-funas-rule* *r* []
roots-rule-list *r* = *add-roots-rule* *r* []
funas-args-rule-list *r* = *add-funas-args-rule* *r* []
<proof>

lemma *[code]*:

vars-trs-list *trs* = *foldr add-vars-rule* *trs* []
funas-trs-list *trs* = *foldr add-funs-rule* *trs* []
funas-trs-list *trs* = *foldr add-funas-rule* *trs* []
funas-args-trs-list *trs* = *foldr add-funas-args-rule* *trs* []
<proof>

definition *insert-vars-trs* :: (*'f*, *'v*) rule list \Rightarrow *'v* list \Rightarrow *'v* list

where

insert-vars-trs *trs* = *foldr insert-vars-rule* *trs*

definition *insert-funs-trs* :: (*'f*, *'v*) rule list \Rightarrow *'f* list \Rightarrow *'f* list

where

insert-funs-trs *trs* = *foldr insert-funs-rule* *trs*

definition *insert-funas-trs* :: (*'f*, *'v*) rule list \Rightarrow (*'f* \times nat) list \Rightarrow (*'f* \times nat) list

where

insert-funas-trs *trs* = *foldr insert-funas-rule* *trs*

definition *insert-roots-trs* :: (*'f*, *'v*) rule list \Rightarrow (*'f* \times nat) list \Rightarrow (*'f* \times nat) list

where

insert-roots-trs *trs* = *foldr insert-roots-rule* *trs*

definition *insert-funas-args-trs* :: (*'f*, *'v*) rule list \Rightarrow (*'f* \times nat) list \Rightarrow (*'f* \times nat) list

where

insert-funas-args-trs *trs* = *foldr insert-funas-args-rule* *trs*

lemma *set-insert-vars-trs* *[simp]*:

set (insert-vars-trs *trs* *xs*) = ($\bigcup r \in \text{set } trs. \text{vars-rule } r$) \cup *set* *xs*
<proof>

lemma *set-insert-funs-trs* *[simp]*:

set (insert-funs-trs *trs* *fs*) = ($\bigcup r \in \text{set } trs. \text{funs-rule } r$) \cup *set* *fs*
<proof>

lemma *set-insert-funas-trs* *[simp]*:

set (insert-funas-trs *trs* *fs*) = ($\bigcup r \in \text{set } trs. \text{funas-rule } r$) \cup *set* *fs*
<proof>

lemma *set-insert-roots-trs* *[simp]*:

$set (insert-roots-trs\ trs\ fs) = (\bigcup r \in set\ trs. roots-rule\ r) \cup set\ fs$
 $\langle proof \rangle$

lemma *set-insert-funas-args-trs* [simp]:
 $set (insert-funas-args-trs\ trs\ fs) = (\bigcup r \in set\ trs. funas-args-rule\ r) \cup set\ fs$
 $\langle proof \rangle$

abbreviation *vars-trs-impl* $trs \equiv insert-vars-trs\ trs\ []$
abbreviation *funas-trs-impl* $trs \equiv insert-funs-trs\ trs\ []$
abbreviation *funas-trs-impl* $trs \equiv insert-funas-trs\ trs\ []$
abbreviation *roots-trs-impl* $trs \equiv insert-roots-trs\ trs\ []$
abbreviation *funas-args-trs-impl* $trs \equiv insert-funas-args-trs\ trs\ []$

definition *defined-list* :: $(f, 'v)\ rule\ list \Rightarrow (f \times nat)\ list$
where
 $defined-list\ R = [the\ (root\ l). (l, r) \leftarrow R, is-Fun\ l]$

lemma *set-defined-list* [simp]:
 $set (defined-list\ R) = \{fn. defined\ (set\ R)\ fn\}$
 $\langle proof \rangle$

definition *check-left-linear-trs* :: $(f :: showl, 'v :: showl)\ rules \Rightarrow showsl\ check$
where

$check-left-linear-trs\ trs =$
 $check-all\ (\lambda r. linear-term\ (fst\ r))\ trs$
 $<+? (\lambda -. showsl-trs\ trs \circ showsl\ (STR\ '(\boxed{\leftarrow})\ is\ not\ left-linear\ (\boxed{\leftarrow}')))$

lemma *check-left-linear-trs* [simp]:
 $isOK\ (check-left-linear-trs\ R) = left-linear-trs\ (set\ R)$
 $\langle proof \rangle$

definition *check-varcond-subset* :: $(-, -)\ rules \Rightarrow showsl\ check$
where

$check-varcond-subset\ R =$
 $check-allm\ (\lambda rule.$
 $check-subseteq\ (vars-term-impl\ (snd\ rule))\ (vars-term-impl\ (fst\ rule))$
 $<+? (\lambda x. showsl\ (STR\ 'free\ variable\ ') \circ showsl\ x$
 $\circ showsl\ (STR\ 'in\ right-hand\ side\ of\ rule\ ') \circ showsl-rule\ rule \circ showsl-nl)$
 $)\ R$

lemma *check-varcond-subset* [simp]:
 $isOK\ (check-varcond-subset\ R) = (\forall\ (l, r) \in set\ R. vars-term\ r \subseteq vars-term\ l)$
 $\langle proof \rangle$

definition *check-varcond-no-Var-lhs* =
 $check-allm\ (\lambda rule.$
 $check\ (is-Fun\ (fst\ rule))$
 $(showsl\ (STR\ 'variable\ left-hand\ side\ in\ rule\ ') \circ showsl-rule\ rule \circ showsl-nl))$

lemma *check-varcond-no-Var-lhs* [simp]:
 $isOK (check-varcond-no-Var-lhs R) \longleftrightarrow (\forall (l, r) \in set R. is-Fun l)$
 <proof>

definition *check-wf-trs* :: $(-, -)$ rules \Rightarrow showsl check
where

check-wf-trs $R = do$ {
check-varcond-no-Var-lhs R ;
check-varcond-subset R
 $\}$ <+? $(\lambda e. showsl (STR "the TRS is not well-formed" \boxed{\leftarrow}) \circ e)$

lemma *check-wf-trs-conf* [simp]:
 $isOK (check-wf-trs R) = wf-trs (set R)$
 <proof>

definition *check-not-wf-trs* :: $(-, -)$ rules \Rightarrow showsl check **where**
check-not-wf-trs $R = check (\neg isOK (check-wf-trs R)) (showsl (STR "The TRS is well formed" \boxed{\leftarrow}))$

lemma *check-not-wf-trs*:
assumes $isOK (check-not-wf-trs R)$
shows $\neg SN (rstep (set R))$
 <proof>

lemma [code]:
 $instance-rule\ lr\ st \longleftrightarrow match-list (\lambda -. fst\ lr) [(fst\ st, fst\ lr), (snd\ st, snd\ lr)] \neq None$
(is $?l = (match-list\ ?d\ ?list \neq None)$
 <proof>

definition
check-CS-subseteq :: $(f, 'v)$ rules $\Rightarrow (f, 'v)$ rules $\Rightarrow (f, 'v)$ rule check
where
check-CS-subseteq $R\ S \equiv check-allm (\lambda (l, r). check (Bex (set S) (instance-rule (l, r))) (l, r)) R$

lemma *check-CS-subseteq* [simp]:
 $isOK (check-CS-subseteq R S) \longleftrightarrow subst.closure (set R) \subseteq subst.closure (set S)$
(is $?l = ?r)$
 <proof>

type-synonym $(f, 'v)rule-map = ((f \times nat) \Rightarrow (f, 'v)rules)option$

fun *computeRuleMapH* :: $(f, 'v)rules \Rightarrow ((f \times nat) \times (f, 'v)rules)list\ option$
where *computeRuleMapH* $\square = Some\ \square$
 $| computeRuleMapH ((Fun\ f\ ts, r) \# rules) = (let\ n = length\ ts\ in\ case\ computeRuleMapH\ rules\ of\ None \Rightarrow None\ | Some\ rm \Rightarrow$

$$\begin{aligned}
& (case\ List.extract\ (\lambda\ (fa,rls). fa = (f,n))\ rm\ of \\
& \quad None \Rightarrow Some\ (((f,n), [(Fun\ f\ ts,r)]) \# rm) \\
& \quad | Some\ (bef,(fa,rls),aft) \Rightarrow Some\ ((fa,(Fun\ f\ ts,r) \# rls) \# bef\ @ \\
& \quad aft))) \\
& | computeRuleMapH\ ((Var\ -, -) \# rules) = None
\end{aligned}$$

definition *computeRuleMap* :: ('f, 'v) rules \Rightarrow ('f, 'v) rule-map **where**

$$\begin{aligned}
computeRuleMap\ rls & \equiv \\
& (case\ computeRuleMapH\ rls\ of \\
& \quad None \Rightarrow None \\
& | Some\ rm \Rightarrow Some\ (\lambda f. \\
& \quad (case\ map-of\ rm\ f\ of \\
& \quad \quad None \Rightarrow [] \\
& \quad | Some\ rls \Rightarrow rls)))
\end{aligned}$$

lemma *computeRuleMapHSound2*: (*computeRuleMapH* *R* = *None*) = ($\exists\ (l, r) \in set\ R. root\ l = None$)
 $\langle proof \rangle$

lemma *computeRuleMapSound2*: (*computeRuleMap* *R* = *None*) = ($\exists\ (l, r) \in set\ R. root\ l = None$)
 $\langle proof \rangle$

lemma *computeRuleMapHSound*: **assumes** *computeRuleMapH* *R* = *Some* *rm*
shows $(\lambda\ (f,rls). (f, set\ rls))\ 'set\ rm = \{((f,n), \{(l,r) \mid l\ r. (l,r) \in set\ R \wedge root\ l = Some\ (f, n)\}) \mid f\ n. \{(l,r) \mid l\ r. (l,r) \in set\ R \wedge root\ l = Some\ (f, n)\} \neq \{\}\} \wedge distinct-eq\ (\lambda\ (f,rls)\ (g,rls'). f = g)\ rm$
 $\langle proof \rangle$

lemma *computeRuleMapSound*:
assumes *computeRuleMap* *R* = *Some* *rm*
shows $(set\ (rm\ (f,n))) = \{(l,r) \mid l\ r. (l,r) \in set\ R \wedge root\ l = Some\ (f, n)\}$
 $\langle proof \rangle$

lemma *computeRuleMap-left-vars*:
shows $(computeRuleMap\ R \neq None) = (\forall\ lr \in set\ R. \forall\ x. fst\ lr \neq Var\ x)$
 $\langle proof \rangle$

lemma *computeRuleMap-defined*: **fixes** *R* :: ('f, 'v) rules
assumes *computeRuleMap* *R* = *Some* *rm*
shows $(rm\ (f,n) = []) = (\neg\ defined\ (set\ R)\ (f,n))$
 $\langle proof \rangle$

lemma *computeRuleMap-None-not-SN*:

assumes *computeRuleMap* $R = \text{None}$
shows $\neg \text{SN-on } (\text{rstep } (\text{set } R)) \{t\}$
 $\langle \text{proof} \rangle$

definition *reverse-rules* $:: ('f, 'v) \text{ rules} \Rightarrow ('f, 'v) \text{ rules}$ **where**
reverse-rules $rs \equiv \text{map prod.swap } rs$

lemma *reverse-rules[simp]*: $\text{set } (\text{reverse-rules } R) = (\text{set } R)^{\wedge -1} \langle \text{proof} \rangle$

definition

map-funs-rules-wa $:: ('f \times \text{nat} \Rightarrow 'g) \Rightarrow ('f, 'v) \text{ rules} \Rightarrow ('g, 'v) \text{ rules}$
where
map-funs-rules-wa $fg R = \text{map } (\lambda(l, r). (\text{map-funs-term-wa } fg l, \text{map-funs-term-wa } fg r)) R$

lemma *map-funs-rules-wa*: $\text{set } (\text{map-funs-rules-wa } fg R) = \text{map-funs-trs-wa } fg (\text{set } R)$
 $\langle \text{proof} \rangle$

lemma *wf-rule* [*code*]:

wf-rule $r \longleftrightarrow$
 $\text{is-Fun } (\text{fst } r) \wedge (\forall x \in \text{set } (\text{vars-term-impl } (\text{snd } r)). x \in \text{set } (\text{vars-term-impl } (\text{fst } r)))$
 $\langle \text{proof} \rangle$

definition *wf-rules-impl* $:: ('f, 'v) \text{ rules} \Rightarrow ('f, 'v) \text{ rules}$
where

wf-rules-impl $R = \text{filter wf-rule } R$

lemma *wf-rules-impl* [*simp*]:

$\text{set } (\text{wf-rules-impl } R) = \text{wf-rules } (\text{set } R)$
 $\langle \text{proof} \rangle$

fun *check-wf-reltrs* $:: (-, -) \text{ rules} \times (-, -) \text{ rules} \Rightarrow \text{showsl check}$ **where**

check-wf-reltrs $(R, S) = (\text{do } \{$
check-wf-trs $R;$
if $R = []$ *then succeed*
else check-varcond-subset S
 $\})$

lemma *check-wf-reltrs*[*simp*]:

$\text{isOK } (\text{check-wf-reltrs } (R, S)) = \text{wf-reltrs } (\text{set } R) (\text{set } S)$
 $\langle \text{proof} \rangle$

declare *check-wf-reltrs.simps*[*simp del*]

definition *check-linear-trs* $:: (-, -) \text{ rules} \Rightarrow \text{showsl check}$ **where**
check-linear-trs $R \equiv$

$check-all (\lambda (l,r). (linear-term\ l) \wedge (linear-term\ r))\ R$
 $<+? (\lambda -. showsl-trs\ R \circ showsl\ (STR\ ' \boxed{\leftarrow} is\ not\ linear\ \boxed{\leftarrow}'))$

lemma *check-linear-trs [simp]:*
 $isOK\ (check-linear-trs\ R) \longleftrightarrow linear-trs\ (set\ R)$
 $\langle proof \rangle$

definition *non-collapsing-impl* $R = list-all\ (is-Fun\ o\ snd)\ R$

lemma *non-collapsing-impl[simp]: non-collapsing-impl* $R = non-collapsing\ (set\ R)$
 $\langle proof \rangle$

type-synonym $('f, 'v)\ term-map = 'f \times nat \Rightarrow ('f, 'v)\ term\ list$

definition *term-map* $:: ('f::compare-order, 'v)\ term\ list \Rightarrow ('f, 'v)\ term-map$ **where**
 $term-map\ ts = fun-of-map\ (rm.\alpha\ (elem-list-to-rm\ (the\ o\ root)\ ts))\ []$

definition
 $is-NF-main :: bool \Rightarrow bool \Rightarrow ('f::compare-order, 'v)\ term-map \Rightarrow ('f, 'v)\ term$
 $\Rightarrow bool$

where

$is-NF-main\ var-cond\ R-empty\ m = (if\ var-cond\ then\ (\lambda -. False)$
 $else\ if\ R-empty\ then\ (\lambda -. True)$
 $else\ (\lambda t. \forall u \in set\ (supteq-list\ t).$
 $if\ is-Fun\ u\ then$
 $\forall l \in set\ (m\ (the\ (root\ u))). \neg\ matches\ u\ l$
 $else\ True))$

lemma *neq-root-no-match:*
assumes $is-Fun\ l$ **and** $the\ (root\ l) \neq the\ (root\ t)$
shows $\neg\ matches\ t\ l$
 $\langle proof \rangle$

lemma *all-not-conv:* $(\forall x \in A. \neg P\ x) = (\neg (\exists x \in A. P\ x))$ $\langle proof \rangle$

lemma *efficient-supteq-list-do-not-match:*
assumes $\forall l \in set\ ls. \forall u \in set\ (supteq-list\ t). the\ (root\ l) \neq the\ (root\ u) \longrightarrow \neg$
 $matches\ u\ l$
shows
 $(\forall l \in set\ ls. \forall u \in set\ (supteq-list\ t). \neg\ matches\ u\ l) \longleftrightarrow$
 $(\forall u \in set\ (supteq-list\ t). \forall l \in set\ (term-map\ ls\ (the\ (root\ u))).$
 $\neg\ matches\ u\ l)$
(is $?lhs \longleftrightarrow ?rhs$ **is** $- \longleftrightarrow (\forall u \in set\ ?subs. \forall l \in set\ (?ls\ u). \neg\ matches\ u\ l))$
 $\langle proof \rangle$

lemma *supteq-list-ex:*
 $(\exists u \in set\ (supteq-list\ l). \exists \sigma. t \cdot \sigma = u) \longleftrightarrow (\exists \sigma. l \succeq t \cdot \sigma)$
 $\langle proof \rangle$

definition *is-NF-trs* $R = \text{is-NF-main } (\exists r \in \text{set } R. \text{ is-Var } (fst\ r))\ (R = [])\ (\text{term-map } (\text{map } fst\ R))$

definition *is-NF-terms* $Q = \text{is-NF-main } (\exists q \in \text{set } Q. \text{ is-Var } q)\ (Q = [])\ (\text{term-map } Q)$

lemma *is-NF-main-NF-trs-conv*:

$\text{is-NF-main } (\exists r \in \text{set } R. \text{ is-Var } (fst\ r))\ (R = [])\ (\text{term-map } (\text{map } fst\ R))\ t \longleftrightarrow$
 $t \in \text{NF-trs } (\text{set } R)$
 $(\text{is is-NF-main ?var ?R ?map } t \longleftrightarrow -)$
 $\langle \text{proof} \rangle$

lemma *is-NF-trs [simp]*:

$\text{is-NF-trs } R = (\lambda\ t. t \in \text{NF-trs } (\text{set } R))$
 $\langle \text{proof} \rangle$

lemma *is-NF-terms [simp]*:

$\text{is-NF-terms } Q = (\lambda\ t. t \in \text{NF-terms } (\text{set } Q))$
 $\langle \text{proof} \rangle$

end

theory *Rewrite-Relations-Impl*

imports

Trs-Impl

Par-Step-Var-Restricted

Multistep

begin

8.4 Implementation of parallel rewriting with variable restriction

context

fixes $R :: ('f, 'v)\text{rules}$ **and** $V :: 'v\ \text{set}$

begin

fun *is-par-rstep-var-restr* $:: ('f, 'v)\ \text{term} \Rightarrow ('f, 'v)\ \text{term} \Rightarrow \text{bool}$

where

$\text{is-par-rstep-var-restr } (\text{Fun } f\ ss)\ (\text{Fun } g\ ts) =$
 $(\text{Fun } f\ ss = \text{Fun } g\ ts \vee$
 $\text{vars-term } (\text{Fun } g\ ts) \cap V = \{\} \wedge (\text{Fun } f\ ss, \text{Fun } g\ ts) \in \text{rrstep } (\text{set } R) \vee$
 $(f = g \wedge \text{length } ss = \text{length } ts \wedge \text{list-all2 is-par-rstep-var-restr } ss\ ts))$
 $| \text{is-par-rstep-var-restr } s\ t = (s = t \vee \text{vars-term } t \cap V = \{\} \wedge (s, t) \in \text{rrstep } (\text{set } R))$

lemma *is-par-rstep-code-helper*: $\text{vars-term } t \cap V = \{\} \longleftrightarrow$

$(\forall\ x \in \text{set } (\text{vars-term-list } t). x \notin V)$

$\langle \text{proof} \rangle$

lemmas *is-par-rstep-var-restr-code*[code] = *is-par-rstep-var-restr.simps*[unfolded *is-par-rstep-code-helper*]

lemma *is-par-rstep-var-restr[simp]*:

is-par-rstep-var-restr s t \longleftrightarrow $(s, t) \in \text{par-rstep-var-restr } (\text{set } R) \ V$
 $\langle \text{proof} \rangle$
end

lemma *par-rstep-var-restr-code*[code-unfold]:

$(s, t) \in \text{par-rstep-var-restr } (\text{set } R) \ V \longleftrightarrow \text{is-par-rstep-var-restr } R \ V \ s \ t$
 $\langle \text{proof} \rangle$

8.5 Implementation of Parallel Rewriting

fun *is-par-rstep* :: $(f, 'v) \text{ rules} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term} \Rightarrow \text{bool}$

where

is-par-rstep *R* (*Fun* *f* *ss*) (*Fun* *g* *ts*) =
 $(\text{Fun } f \text{ ss} = \text{Fun } g \text{ ts} \vee (\text{Fun } f \text{ ss}, \text{Fun } g \text{ ts}) \in \text{rrstep } (\text{set } R) \vee$
 $(f = g \wedge \text{length } ss = \text{length } ts \wedge \text{list-all2 } (\text{is-par-rstep } R) \text{ ss ts}))$
 $| \text{is-par-rstep } R \ s \ t = (s = t \vee (s, t) \in \text{rrstep } (\text{set } R))$

lemma *is-par-rstep[simp]*:

is-par-rstep *R* *s* *t* $\longleftrightarrow (s, t) \in \text{par-rstep } (\text{set } R)$
 $\langle \text{proof} \rangle$

lemma *par-rstep-code*[code-unfold]: $(s, t) \in \text{par-rstep } (\text{set } R) \longleftrightarrow \text{is-par-rstep } R \ s \ t$
 $\langle \text{proof} \rangle$

8.6 Generate all root-rewrites (for well-formed TRS)

fun *root-rewrite* :: $(f, 'v) \text{ rules} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term list}$

where

root-rewrite *R* *s* = *concat* (*map* $(\lambda (l, r).$
 $(\text{case match } s \text{ l of}$
 $\text{None} \Rightarrow []$
 $| \text{Some } \sigma \Rightarrow [(r \cdot \sigma)])) \ R$

lemma *root-rewrite-sound*:

assumes $t \in \text{set } (\text{root-rewrite } R \ s)$
shows $(s, t) \in \text{rrstep } (\text{set } R)$
 $\langle \text{proof} \rangle$

8.7 Generate all parallel rewrites (for well formed TRS)

fun *parallel-rewrite* :: $(f, 'v) \text{ rules} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term list}$

where

parallel-rewrite *R* (*Var* *x*) = [*Var* *x*]
 $| \text{parallel-rewrite } R \ (\text{Fun } f \text{ ss}) = \text{remdups}$
 $(\text{root-rewrite } R \ (\text{Fun } f \text{ ss}) @ \text{map } (\lambda ss. \text{Fun } f \text{ ss}) (\text{product-lists } (\text{map } (\text{parallel-rewrite } R) \text{ ss})))$

lemma *parallel-rewrite-par-step*:

assumes $t \in \text{set } (\text{parallel-rewrite } R \ s)$

shows $(s, t) \in \text{par-rstep } (\text{set } R)$

<proof>

fun *root-steps-substs* :: $(f, 'v) \text{ rules} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term} \Rightarrow ((f, 'v) \text{ term list} \times (f, 'v) \text{ term list}) \text{ list}$

where

root-steps-substs $R \ s \ t = \text{concat } (\text{map } (\lambda (l, r).$

$(\text{case match } s \ l \ \text{of}$

$\text{None} \Rightarrow []$

$| \text{Some } \sigma \Rightarrow (\text{case match } t \ r \ \text{of}$

$\text{None} \Rightarrow []$

$| \text{Some } \tau \Rightarrow (\text{let var-list} = \text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l) \text{ in}$

$[(\text{map } \sigma \ \text{var-list}, \text{map } \tau \ \text{var-list})])))$

$R)$

lemma *root-steps-substs-exists*:

assumes $(ss, ts) \in \text{set } (\text{root-steps-substs } R \ s \ t)$

shows $\exists l \ r \ \sigma \ \tau \ vl. (l, r) \in \text{set } R \wedge vl = \text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l) \wedge$

$l \cdot \sigma = s \wedge r \cdot \tau = t \wedge (ss, ts) = (\text{map } \sigma \ vl, \text{map } \tau \ vl)$

<proof>

lemma *size-match-subst-Fun*:

assumes *is-Fun* l **and** $x \in \text{vars-term } l$

and *match*: $\text{match } s \ l = \text{Some } \tau$

shows $\text{size } (\tau \ x) < \text{size } s$

<proof>

8.8 Implementation of Multistep Rewriting

abbreviation *remove-trivial-rules* $R \equiv \text{filter } (\lambda (l, r). \neg (\text{is-Var } l) \vee \neg (\text{is-Var } r)) \ R$

lemma *trivial-rrstep*:

assumes $\exists x \ y. (\text{Var } x, \text{Var } y) \in R \wedge x \neq y$

shows $(s, t) \in \text{rrstep } R$

<proof>

lemma *size-root-steps-substs*:

assumes $(ss, ts) \in \text{set } (\text{root-steps-substs } (\text{remove-trivial-rules } R) \ s \ t)$

and $s' \in \text{set } ss \ t' \in \text{set } ts$

shows $\text{size } s' + \text{size } t' < \text{size } s + \text{size } t$

<proof>

function *(sequential) is-mstep* :: $(f, 'v) \text{ rules} \Rightarrow (f, 'v) \text{ term} \Rightarrow (f, 'v) \text{ term} \Rightarrow$

bool
where
 $is_mstep\ R\ (Fun\ f\ ss)\ (Fun\ g\ ts) =$
 $(Fun\ f\ ss = Fun\ g\ ts \vee (Fun\ f\ ss, Fun\ g\ ts) \in rrstep\ (set\ R) \vee$
 $list_ex\ (\lambda\ (ss, ts). list_all2\ (is_mstep\ R)\ ss\ ts)\ (root_steps_subst\ (remove_trivial_rules$
 $R)\ (Fun\ f\ ss)\ (Fun\ g\ ts))) \vee$
 $(f = g \wedge length\ ss = length\ ts \wedge list_all2\ (is_mstep\ R)\ ss\ ts))$
 $| is_mstep\ R\ s\ t = (s = t \vee (s, t) \in rrstep\ (set\ R) \vee$
 $list_ex\ (\lambda\ (ss, ts). list_all2\ (is_mstep\ R)\ ss\ ts)\ (root_steps_subst\ (remove_trivial_rules$
 $R)\ s\ t))$
 $\langle proof \rangle$

termination $\langle proof \rangle$

Show that all multisteps are covered by the definition above.

lemma *mstep-is-mstep*:
assumes $(s, t) \in mstep\ (set\ R)$
shows $is_mstep\ R\ s\ t$
 $\langle proof \rangle$

lemma *mstep-root-helper*:
assumes $list_ex\ (\lambda\ (ss, ts). list_all2\ (is_mstep\ R)\ ss\ ts)\ (root_steps_subst\ (remove_trivial_rules$
 $R)\ s\ t)$
and $\bigwedge\ ss\ ts\ s'\ t'. (ss, ts) \in set\ (root_steps_subst\ (remove_trivial_rules\ R)\ s\ t)$
 $\implies s' \in set\ ss \implies t' \in set\ ts \implies is_mstep\ R\ s'\ t' \implies (s', t') \in mstep\ (set\ R)$
shows $(s, t) \in mstep\ (set\ R)$
 $\langle proof \rangle$

lemma *is-mstep-mstep*:
assumes $is_mstep\ R\ s\ t$
shows $(s, t) \in mstep\ (set\ R)$
 $\langle proof \rangle$

lemma *is-mstep[simp]*:
 $is_mstep\ R\ s\ t \iff (s, t) \in mstep\ (set\ R)$
 $\langle proof \rangle$

lemma *mstep-code[code-unfold]*: $(s, t) \in mstep\ (set\ R) \iff is_mstep\ R\ s\ t\ \langle proof \rangle$

fun *root-subst-with-rhs* :: $('f, 'v)\ rules \Rightarrow ('f, 'v)\ term \Rightarrow (('f, 'v)\ term \times ('f, 'v)$
 $term\ list)\ list$

where
 $root_subst_with_rhs\ R\ s = concat\ (map\ (\lambda\ (l, r).$
 $(case\ match\ s\ l\ of$
 $None \Rightarrow []$
 $| Some\ \sigma \Rightarrow [(r, map\ \sigma\ (vars_distinct\ r))]))$

R)

lemma *root-steps-subst-rhs-exists*:

assumes $(r, ss) \in \text{set } (\text{root-subst-with-rhs } R \ s)$

shows $\exists l \ \sigma. (l, r) \in \text{set } R \wedge l \cdot \sigma = s \wedge ss = \text{map } \sigma \ (\text{vars-distinct } r)$

$\langle \text{proof} \rangle$

context

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

assumes $\text{wf-trs } (\text{set } R)$

begin

private lemma $*$: $\text{list-all } (\lambda(l, r). \text{is-Fun } l \wedge (\text{vars-term } r \subseteq \text{vars-term } l)) \ R$

$\langle \text{proof} \rangle$

lemma *varcond*:

$\bigwedge l \ r. (l, r) \in \text{set } R \implies \text{is-Fun } l \wedge \text{vars-term } r \subseteq \text{vars-term } l$

$\langle \text{proof} \rangle$

lemma [*termination-simp*]:

assumes $(l, r) \in \text{set } R$

and $\text{Some } \sigma = \text{match } (\text{Fun } g \ ts) \ l$

and $x \in \text{vars-term } r$

shows $\text{size } (\sigma \ x) < \text{Suc } (\text{size-list size } ts)$

$\langle \text{proof} \rangle$

fun *mstep-rewrite-main* :: $('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term list}$

where

$\text{mstep-rewrite-main } (\text{Var } x) = [\text{Var } x]$

$| \text{mstep-rewrite-main } (\text{Fun } f \ ss) = \text{remdups } ($

$\text{concat } (\text{map } (\lambda(r, ts).$

$\text{map } (\lambda \text{args}. r \cdot (\text{mk-subst Var } (\text{zip } (\text{vars-distinct } r) \ \text{args}))) \ (\text{product-lists}$

$(\text{map } \text{mstep-rewrite-main } ts))))$

$(\text{root-subst-with-rhs } R \ (\text{Fun } f \ ss))))$

$\text{@}(\text{map } (\lambda ss. \text{Fun } f \ ss) \ (\text{product-lists } (\text{map } \text{mstep-rewrite-main } ss))))$

lemma *mstep-rewrite-main-mstep*:

assumes $t \in \text{set } (\text{mstep-rewrite-main } s)$

shows $(s, t) \in \text{mstep } (\text{set } R)$

$\langle \text{proof} \rangle$

end

We need to be able to export code for *mstep-rewrite-main*, hence the following definitions (adapted from template by Rene).

typedef $('f, 'v) \text{ wfTRS} = \{R :: ('f, 'v) \text{ rules}. \text{wf-trs } (\text{set } R)\}$

$\langle \text{proof} \rangle$

setup-lifting *type-definition-wfTRS*

lift-definition *get-TRS* :: ('f, 'v) wfTRS \Rightarrow ('f, 'v) rules **is** $\lambda R. R$ *<proof>*

lemma *is-wf-get-TRS*: wf-trs (set (get-TRS R'))
<proof>

definition *mstep-rewrite-wf* R = *mstep-rewrite-main* (get-TRS R)

lemmas *mstep-rewrite-wf-simps* = *mstep-rewrite-main.simps*[OF *is-wf-get-TRS*,
folded mstep-rewrite-wf-def]

declare *mstep-rewrite-wf-simps*[code]

lift-definition (code-dt) *get-wfTRS* :: ('f :: showl, 'v :: showl) rules \Rightarrow ('f, 'v)
wfTRS option is
 $\lambda R. \text{if } \text{isOK } (\text{check-wf-trs } R) \text{ then } \text{Some } R \text{ else } \text{None}$
<proof>

definition *err-wf* **where** *err-wf* = STR "TRS is not well-formed"

definition *mstep-dummy-impl* R s t = ((s,t) \in *mstep* (set R))

lemma *mstep-dummy-impl*[code]: *mstep-dummy-impl* R = Code.abort (STR "mstep-dummy")
($\lambda -. \text{mstep-dummy-impl } R$)
<proof>

lift-definition (code-dt) *get-wfTRS-sub* :: ('f :: showl, 'v :: showl) rules \Rightarrow ('f, 'v)
wfTRS is
 $\lambda R. \text{if } \text{isOK } (\text{check-wf-trs } R) \text{ then } R \text{ else } \text{Code.abort } \text{err-wf } (\lambda -. \square)$
<proof>

definition *mstep-rewrite* R = *mstep-rewrite-wf* (get-wfTRS-sub R)

lemma *mstep-rewrite-mstep*:
assumes $t \in \text{set } (\text{mstep-rewrite } R \ s)$
shows $(s, t) \in \text{mstep } (\text{set } R)$
<proof>

end

9 A Concrete Unification Algorithm

theory *Unification-More*

imports

First-Order-Terms.Unification

First-Order-Rewriting.Term-Impl

begin

lemma *set-subst-list* [*simp*]:
 $set (subst-list \sigma E) = subst-set \sigma (set E)$
 $\langle proof \rangle$

lemma *mgu-var-disjoint-right*:
fixes $s\ t :: ('f, 'v) term$ **and** $\sigma\ \tau :: ('f, 'v) subst$ **and** T
assumes $s: vars-term\ s \subseteq S$
and $inj: inj\ T$
and $ST: S \cap range\ T = \{\}$
and $id: s \cdot \sigma = t \cdot \tau$
shows $\exists\ \mu\ \delta. mgu\ s\ (map-vars-term\ T\ t) = Some\ \mu \wedge$
 $s \cdot \sigma = s \cdot \mu \cdot \delta \wedge$
 $(\forall t::('f, 'v) term. t \cdot \tau = map-vars-term\ T\ t \cdot \mu \cdot \delta) \wedge$
 $(\forall x \in S. \sigma\ x = \mu\ x \cdot \delta)$
 $\langle proof \rangle$

abbreviation (*input*) $x-var :: string \Rightarrow string$ **where** $x-var \equiv Cons\ (CHR\ "x")$
abbreviation (*input*) $y-var :: string \Rightarrow string$ **where** $y-var \equiv Cons\ (CHR\ "y")$
abbreviation (*input*) $z-var :: string \Rightarrow string$ **where** $z-var \equiv Cons\ (CHR\ "z")$

lemma *mgu-var-disjoint-right-string*:
fixes $s\ t :: ('f, string) term$ **and** $\sigma\ \tau :: ('f, string) subst$
assumes $s: vars-term\ s \subseteq range\ x-var \cup range\ z-var$
and $id: s \cdot \sigma = t \cdot \tau$
shows $\exists\ \mu\ \delta. mgu\ s\ (map-vars-term\ y-var\ t) = Some\ \mu \wedge$
 $s \cdot \sigma = s \cdot \mu \cdot \delta \wedge (\forall t::('f, string) term. t \cdot \tau = map-vars-term\ y-var\ t \cdot \mu \cdot \delta)$
 \wedge
 $(\forall x \in range\ x-var \cup range\ z-var. \sigma\ x = \mu\ x \cdot \delta)$
 $\langle proof \rangle$

lemma *not-elem-subst-of*:
assumes $x \notin set\ (map\ fst\ xs)$
shows $(subst-of\ xs)\ x = Var\ x$
 $\langle proof \rangle$

lemma *subst-of-id*:
assumes $\bigwedge s. s \in (set\ ss) \longrightarrow (\exists x\ t. s = (x, t) \wedge t = Var\ x)$
shows $subst-of\ ss = Var$
 $\langle proof \rangle$

lemma *subst-of-apply*:
assumes $(x, t) \in set\ ss$
and $\forall (y, s) \in set\ ss. (y = x \longrightarrow s = t)$
and $set\ (map\ fst\ ss) \cap vars-term\ t = \{\}$
shows $subst-of\ ss\ x = t$
 $\langle proof \rangle$

lemma *unify-equation-same*:

assumes $\text{fst } e = \text{snd } e$

shows $\text{unify } (E1 @ e \# E2) \text{ } ys = \text{unify } (E1 @ E2) \text{ } ys$

<proof>

lemma *unify-filter-same*:

shows $\text{unify } (\text{filter } (\lambda e. \text{fst } e \neq \text{snd } e) \text{ } E) \text{ } ys = \text{unify } E \text{ } ys$

<proof>

lemma *unify-ctxt-same*:

shows $\text{unify } ((C \langle s \rangle, C \langle t \rangle) \# xs) \text{ } ys = \text{unify } ((s, t) \# xs) \text{ } ys$

<proof>

10 Unification of linear and variable disjoint terms

definition *left-substs* :: $(f, 'v) \text{ term} \Rightarrow (f, 'w) \text{ term} \Rightarrow ('v \times (f, 'w) \text{ term}) \text{ list}$

where $\text{left-substs } s \text{ } t = (\text{let } \text{filtered-vars} = \text{filter } (\lambda(-, p). p \in \text{poss } t) (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } t)))$
 $\text{in } \text{map } (\lambda(x, p). (x, t|-p)) \text{ filtered-vars}$

definition *right-substs* :: $(f, 'v) \text{ term} \Rightarrow (f, 'w) \text{ term} \Rightarrow ('w \times (f, 'v) \text{ term}) \text{ list}$

where $\text{right-substs } s \text{ } t = (\text{let } \text{filtered-vars} = \text{filter } (\lambda(-, q). q \in \text{fun-poss } s) (\text{zip } (\text{vars-term-list } t) (\text{var-poss-list } t)))$
 $\text{in } \text{map } (\lambda(y, q). (y, s|-q)) \text{ filtered-vars}$

abbreviation *linear-unifier* $s \text{ } t \equiv \text{subst-of } ((\text{left-substs } s \text{ } t) @ (\text{right-substs } s \text{ } t))$

lemma *left-substs-imp-props*:

assumes $(x, u) \in \text{set } (\text{left-substs } s \text{ } t)$

shows $\exists p. p \in \text{poss } s \wedge s|-p = \text{Var } x \wedge p \in \text{poss } t \wedge t|-p = u$

<proof>

lemma *props-imp-left-substs*:

assumes $p \in \text{poss } s$ **and** $s|-p = \text{Var } x$ **and** $p \in \text{poss } t$ **and** $t|-p = u$

shows $(x, u) \in \text{set } (\text{left-substs } s \text{ } t)$

<proof>

lemma *right-substs-imp-props*:

assumes $(x, u) \in \text{set } (\text{right-substs } s \text{ } t)$

shows $\exists q. q \in \text{fun-poss } s \wedge s|-q = u \wedge q \in \text{poss } t \wedge t|-q = \text{Var } x$

<proof>

lemma *props-imp-right-substs*:

assumes $q \in \text{fun-poss } s$ **and** $s|-q = u$ **and** $q \in \text{poss } t$ **and** $t|-q = \text{Var } x$

shows $(x, u) \in \text{set } (\text{right-substs } s \text{ } t)$

<proof>

lemma *map-fst-left-substs*:

$\text{set } (\text{map fst } (\text{left-substs } s \ t)) \subseteq \text{vars-term } s$
 $\langle \text{proof} \rangle$

lemma *map-snd-left-substs*:
 assumes $t' \in \text{set } (\text{map snd } (\text{left-substs } s \ t))$
 shows $\text{vars-term } t' \subseteq \text{vars-term } t$
 $\langle \text{proof} \rangle$

lemma *map-fst-right-substs*:
 $\text{set } (\text{map fst } (\text{right-substs } s \ t)) \subseteq \text{vars-term } t$
 $\langle \text{proof} \rangle$

lemma *map-snd-right-substs*:
 assumes $t' \in \text{set } (\text{map snd } (\text{right-substs } s \ t))$
 shows $\text{vars-term } t' \subseteq \text{vars-term } s$
 $\langle \text{proof} \rangle$

lemma *distinct-map-fst-left-substs*:
 assumes *linear-term* t
 shows *distinct* $(\text{map fst } (\text{left-substs } t \ s))$
 $\langle \text{proof} \rangle$

lemma *distinct-map-fst-right-substs*:
 assumes *linear-term* t
 shows *distinct* $(\text{map fst } (\text{right-substs } s \ t))$
 $\langle \text{proof} \rangle$

lemma *is-partition-map-snd-left-substs*:
 assumes *linear-term* s *linear-term* t
 shows *is-partition* $(\text{map } (\text{vars-term } \circ \text{snd}) (\text{left-substs } t \ s))$
 $\langle \text{proof} \rangle$

lemma *is-partition-map-snd-right-substs*:
 assumes *linear-term* s *linear-term* t
 shows *is-partition* $(\text{map } (\text{vars-term } \circ \text{snd}) (\text{right-substs } t \ s))$
 $\langle \text{proof} \rangle$

lemma *distinct-fst-lsubsts-snd-rsubsts*:
 assumes *linear-term* s
 shows $(\text{set } (\text{map fst } (\text{left-substs } s \ t))) \cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{right-substs } s \ t))) = \{\}$
 $\langle \text{proof} \rangle$

lemma *distinct-fst-rsubsts-snd-lsubsts*:
 assumes *linear-term* t
 shows $(\text{set } (\text{map fst } (\text{right-substs } s \ t))) \cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{left-substs } s \ t))) = \{\}$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-same*:
shows $(\text{linear-unifier } t \ t) = \text{Var}$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-var1*:
shows $\text{linear-unifier } (\text{Var } x) \ t = \text{subst } x \ t$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-var2*:
shows $\text{linear-unifier } (\text{Fun } f \ ts) \ (\text{Var } x) = \text{subst } x \ (\text{Fun } f \ ts)$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-id*:
assumes $x \notin \text{vars-term } s$ **and** $x \notin \text{vars-term } t$
shows $(\text{linear-unifier } s \ t) \ x = \text{Var } x$
 $\langle \text{proof} \rangle$

lemma *vars-subst-of*:
 $\text{vars-subst } (\text{subst-of } ts) \subseteq \text{set } (\text{map } \text{fst } ts) \cup \bigcup (\text{set } (\text{map } (\text{vars-term} \circ \text{snd}) \ ts))$
 $\langle \text{proof} \rangle$

lemma *vars-subst-linear-unifier*: $\text{vars-subst } (\text{linear-unifier } s \ t) \subseteq \text{vars-term } s \cup \text{vars-term } t$
 $\langle \text{proof} \rangle$

lemma *decompose-is-partition-vars-subst*:
assumes $\text{lin}:\text{linear-term } (\text{Fun } f \ ss) \ \text{linear-term } (\text{Fun } g \ ts)$
and $\text{disj}:\text{vars-term } (\text{Fun } f \ ss) \cap \text{vars-term } (\text{Fun } g \ ts) = \{\}$
and $\text{ds}:\text{decompose } (\text{Fun } f \ ss) \ (\text{Fun } g \ ts) = \text{Some } ds$
shows $\text{is-partition } (\text{map } \text{vars-subst } (\text{map } (\lambda(s,t). \text{linear-unifier } s \ t) \ ds))$
 $\langle \text{proof} \rangle$

lemma *compose-exists-subst*:
assumes $\text{compose } \sigma \ s \ x \neq \text{Var } x$
shows $\exists i < \text{length } \sigma s. (\forall j < i. (\sigma s!j) \ x = \text{Var } x) \wedge (\sigma s!i) \ x \neq \text{Var } x$
 $\langle \text{proof} \rangle$

lemma *subst-of-exists-binding*:
assumes $\text{subst-of } xs \ y \neq \text{Var } y$
shows $\exists i < \text{length } xs. \text{fst } (xs!i) = y \wedge (\forall x \in \text{set } (\text{drop } (i+1) \ xs). \text{fst } x \neq y)$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-obtain-binding*:
assumes $\text{disj}:\text{vars-term } s \cap \text{vars-term } t = \{\}$ **and** $\text{lin-}s:\text{linear-term } s$ **and** $\text{lin-}t:\text{linear-term } t$
and $u:(\text{linear-unifier } s \ t) \ x = u \ u \neq \text{Var } x$
shows $(x \in \text{vars-term } s \wedge (x,u) \in \text{set } (\text{left-substs } s \ t)) \vee (x \in \text{vars-term } t \wedge (x,u) \in \text{set } (\text{right-substs } s \ t))$
 $\langle \text{proof} \rangle$

connection between *left-substs* and *right-substs* and decomposition of functions

lemma *decompose-left-substs*:

assumes *decompose* (*Fun f ss*) (*Fun g ts*) = *Some ds*
shows *set* (*left-substs* (*Fun f ss*) (*Fun g ts*)) = ($\bigcup_{e \in \text{set } ds} \text{set } (\text{left-substs } (\text{fst } e) (\text{snd } e))$) (**is** ?*left* = ?*right*)
 $\langle \text{proof} \rangle$

lemma *decompose-right-substs*:

assumes *decompose* (*Fun f ss*) (*Fun g ts*) = *Some ds*
shows *set* (*right-substs* (*Fun f ss*) (*Fun g ts*)) = ($\bigcup_{e \in \text{set } ds} \text{set } (\text{right-substs } (\text{fst } e) (\text{snd } e))$) (**is** ?*left* = ?*right*)
 $\langle \text{proof} \rangle$

lemma *subst-compose-id*:

assumes $\bigwedge \tau. \tau \in \text{set } \tau s \implies t \cdot \tau = t$
shows $t \cdot (\text{compose } \tau s) = t$
 $\langle \text{proof} \rangle$

lemma *subst-compose-distinct-vars*:

assumes $\sigma = \text{compose } \tau s$ **and** *part:is-partition* (*map vars-subst* τs)
and $\tau i : \tau i \in \text{set } \tau s$ **and** $s : \tau i x = s \ s \neq \text{Var } x$
shows $\sigma x = s$
 $\langle \text{proof} \rangle$

lemma *subst-id-compose*:

assumes $\sigma = \text{compose } \tau s$ **and** *part:is-partition* (*map vars-subst* τs)
and $t \cdot \sigma = t$
and $\tau \in \text{set } \tau s$
shows $t \cdot \tau = t$
 $\langle \text{proof} \rangle$

lemma *compose-subst-of*:

assumes $\text{set } ss = \bigcup (\text{set } ' \text{set } ss')$
and *is-partition* (*map* (*vars-term* $\circ \text{snd}$) *ss*) **and** *distinct* (*map fst ss*)
and $\text{set } (\text{map fst } ss) \cap \bigcup (\text{set } (\text{map } (\text{vars-term} \circ \text{snd}) ss)) = \{\}$
and *is-partition* (*map vars-subst* (*map subst-of* ss'))
shows $\text{subst-of } ss = \text{compose } (\text{map subst-of } ss')$ (**is** ? $\sigma = ?\tau$)
 $\langle \text{proof} \rangle$

lemma *linear-term-decompose-subst-id*:

assumes *lin:linear-term* (*Fun f ss*) *linear-term* (*Fun g ts*)
and *disj:vars-term* (*Fun f ss*) \cap *vars-term* (*Fun g ts*) = $\{\}$
and *decompose* (*Fun f ss*) (*Fun g ts*) = *Some ds*
and $i : i < \text{length } ds$ **and** $\sigma : \sigma = \text{linear-unifier } (\text{fst } (ds!i)) (\text{snd } (ds!i))$
and $j : j < \text{length } ds \ j \neq i$
shows $\text{fst } (ds!j) \cdot \sigma = \text{fst } (ds!j) \wedge \text{snd } (ds!j) \cdot \sigma = \text{snd } (ds!j)$
 $\langle \text{proof} \rangle$

lemma *linear-unifier-decompose*:
assumes *linear-term* (*Fun* *f ss*) *linear-term* (*Fun* *g ts*)
and *disj:vars-term* (*Fun* *f ss*) \cap *vars-term* (*Fun* *g ts*) = {}
and *ds:decompose* (*Fun* *f ss*) (*Fun* *g ts*) = *Some ds*
shows *linear-unifier* (*Fun* *f ss*) (*Fun* *g ts*) = *compose* (*map* ($\lambda(s,t).$ *linear-unifier* *s t*) *ds*)
<proof>

lemma *unify-linear-terms*:
assumes *unify es substs* = *Some res*
and *compose* (*subst-of substs* $\#$ (*map* ($\lambda(s,t).$ *linear-unifier s t*) *es*)) = τ
and $\forall t \in \text{set } (\text{map } \text{fst } \text{es}) \cup \text{set } (\text{map } \text{snd } \text{es}). \text{linear-term } t$
and $\bigwedge i\ j\ \sigma. i < j \implies j < \text{length } \text{es} \implies \sigma = \text{linear-unifier } (\text{fst } (\text{es}!i)) (\text{snd } (\text{es}!i)) \implies$
 $(\text{fst } (\text{es}!j)) \cdot \sigma = \text{fst } (\text{es}!j) \wedge (\text{snd } (\text{es}!j)) \cdot \sigma = \text{snd } (\text{es}!j)$
and $\bigwedge i. i < \text{length } \text{es} \implies \text{vars-term } (\text{fst } (\text{es}!i)) \cap \text{vars-term } (\text{snd } (\text{es}!i)) = \{\}$
shows *subst-of res* = τ
<proof>

lemma *mgu-distinct-vars-term-list*:
assumes *unif:unifiers* $\{(s, t)\} \neq \{\}$
and *distinct:distinct* (*(vars-term-list s) @ (vars-term-list t)*)
shows *mgu s t* = *Some (linear-unifier s t)*
<proof>

end

11 Sets of Unifiers

theory *Unifiers-More*
imports
First-Order-Terms.Term-More
First-Order-Terms.Unifiers
begin

lemma *is-mguI*:
fixes $\sigma :: ('f, 'v) \text{subst}$
assumes $\forall (s, t) \in E. s \cdot \sigma = t \cdot \sigma$
and $\bigwedge \tau :: ('f, 'v) \text{subst}. \forall (s, t) \in E. s \cdot \tau = t \cdot \tau \implies \exists \gamma :: ('f, 'v) \text{subst}. \tau = \sigma \circ_s \gamma$
shows *is-mgu* σE
<proof>

lemma *subst-set-insert [simp]*:
 $\text{subst-set } \sigma (\text{insert } e E) = \text{insert } (\text{fst } e \cdot \sigma, \text{snd } e \cdot \sigma) (\text{subst-set } \sigma E)$
<proof>

lemma *unifiable-UnD [dest]*:

$unifiable (M \cup N) \implies unifiable M \wedge unifiable N$
 $\langle proof \rangle$

lemma *supt-imp-not-unifiable*:

assumes $s \triangleright t$

shows $\neg unifiable \{(t, s)\}$

$\langle proof \rangle$

lemma *unifiable-insert-Var-swap* [simp]:

$unifiable (insert (t, Var x) E) \longleftrightarrow unifiable (insert (Var x, t) E)$

$\langle proof \rangle$

lemma *unifiers-Int1* [simp]:

$(s, t) \in E \implies unifiers \{(s, t)\} \cap unifiers E = unifiers E$

$\langle proof \rangle$

lemma *imgu-linear-var-disjoint*:

assumes $is-imgu \sigma \{(l2 \mid - p, l1)\}$

and $p \in poss \ l2$

and *linear-term* $l2$

and $vars-term \ l1 \cap vars-term \ l2 = \{\}$

and $q \in poss \ l2$

and *parallel-pos* $p \ q$

shows $l2 \mid - q = l2 \mid - q \cdot \sigma$

$\langle proof \rangle$

end