

First-Order Rewriting

TODO

December 18, 2024

Contents

1	Closure-Operations on Relations	4
2	Term Rewriting	6
3	Term Rewrite Systems	8
3.1	Variables of Rules	8
3.2	Variables of TRSs	8
3.3	Function Symbols of Rules	9
3.4	Function Symbols of TRSs	9
3.5	Rewrite steps at a fixed position	83
3.6	Parallel rewrite relation	85
3.7	Function Symbols and Variables	92
3.8	Function Symbols and Variables	92
4	Overloading for the Sharp Symbol	102
4.1	Q-Restricted Rewriting	109
5	The Critical Pair Lemma (with variables fixed to type string)	180
6	Preliminaries	193
6.1	Combinators	198
6.2	Distinct Lists and Partitions	199
7	Multihole Contexts	202
7.1	Partitioning lists into chunks of given length	202
7.2	Conversions from and to multihole contexts	206
7.3	Semilattice Structures	244
7.4	All positions of a multi-hole context	265
7.5	More operations on multi-hole contexts	268
7.6	An inverse of <i>fill-holes</i>	273
7.7	Ditto for <i>fill-holes-mctxt</i>	275
7.8	Function symbols of prefixes	277
7.9	Parallel Rewriting using Multihole Contexts	277

8 Multi-Step Rewriting	282
9 Permutations	294
10 Permutation Types	297
11 Pure Types	312
12 Variables and Variable Positions	327
12.1 Useful abstractions	327
12.2 Additional Functions on Terms	334
13 A Concrete Unification Algorithm	348
14 Unification of linear and variable disjoint terms	353
15 Sets of Unifiers	375
16 Renaming of Terms, Substitutions, TRSs, ...	377
16.1 Output	405
16.2 Implementation of parallel rewriting with variable restriction	424
16.3 Implementation of Parallel Rewriting	427
16.4 Generate all root-rewrites (for well-formed TRS)	428
16.5 Generate all parallel rewrites (for well formed TRS)	428
16.6 Implementation of Multistep Rewriting	430

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

proof (*induct rule: compute-NF.raw-induct[OF - res, of $\lambda g a b. g = f \longrightarrow (a,b) \in r^*$, THEN mp[OF - refl]]*)

case (*1 cnf g a b*)

show *?case*

proof

assume *g = f*

note *1 = 1[unfolded this]*

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

proof (*cases f a*)

case *None*

```

    with 1(2) show ?thesis by simp
  next
    case (Some c)
    from 1(2)[unfolded this] have cnf f c = Some b by simp
    from 1(1)[OF this] have (c,b) ∈ r∧* by auto
    with f-sound[OF Some] show ?thesis by auto
  qed
qed
qed

lemma compute-NF-complete: assumes res: compute-NF f a = Some b
  and f-complete:  $\bigwedge a. f a = \text{None} \implies a \in NF\ r$ 
  shows  $b \in NF\ r$ 
proof (induct rule: compute-NF.raw-induct[OF - res, of  $\lambda g\ a\ b. g = f \longrightarrow b \in NF\ r$ , THEN mp[OF - refl]])
  case (1 cnf g a b)
  show ?case
  proof
    assume g = f
    note 1 = 1[unfolded this]
    show  $b \in NF\ r$ 
    proof (cases f a)
      case None
      with f-complete[OF None] 1(2)
      show ?thesis by simp
    next
      case (Some c)
      from 1(2)[unfolded this] have cnf f c = Some b by simp
      from 1(1)[OF this] show ?thesis by simp
    qed
  qed
qed
qed

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)
proof -
  let ?r = {(a,b). f a = Some b}
  have ?r ⊆ r using f-sound by auto
  from SN-subset[OF SN this] have SNr: SN ?r .
  show ?thesis
  proof (induct rule: SN-induct[OF SNr, of  $\lambda a. ?P\ a$ ])
    case (1 a)
    show ?case
    proof (cases f a)
      case None then show ?thesis by auto
    next
      case (Some b)
      then have (a,b) ∈ ?r by simp
    qed
  qed

```

```

    from 1[OF this] f-sound[OF Some] show ?thesis
    using Some by auto
  qed
qed
qed

definition compute-trancl A R =  $R^+ \text{ “ } A$ 
lemma compute-trancl-rtrancl[code-unfold]:  $\{b. (a,b) \in R^*\} = \text{insert } a \text{ (compute-trancl } \{a\} R)$ 
proof –
  have id:  $R^* = (Id \cup R^+)$  by regexp
  show ?thesis unfolding id compute-trancl-def by auto
qed

lemma [code]: compute-trancl A R = (let B =  $R \text{ “ } A$  in
  if  $B \subseteq \{\}$  then  $\{\}$  else  $B \cup \text{compute-trancl } B \{ ab \in R . \text{fst } ab \notin A \wedge \text{snd } ab \notin B \}$ )
proof –
  have  $R: R^+ = R \circ R^*$  by regexp
  define B where  $B = R \text{ “ } A$ 
  define R' where  $R' = \{ab \in R. \text{fst } ab \notin A \wedge \text{snd } ab \notin B\}$ 
  note d = compute-trancl-def
  show ?thesis unfolding Let-def B-def[symmetric] R'-def[symmetric] d
  proof (cases  $B \subseteq \{\}$ )
    case True
    then show  $R^+ \text{ “ } A = (\text{if } B \subseteq \{\} \text{ then } \{\} \text{ else } B \cup R'^+ \text{ “ } B)$  unfolding
B-def R by auto
  next
    case False
    have  $R' \subseteq R$  unfolding R'-def by auto
    then have  $R'^+ \subseteq R^+$  by (rule trancl-mono-set)
    also have  $\dots \subseteq R^*$  by auto
    finally have mono:  $R'^+ \subseteq R^*$  .
    have  $B \cup R'^+ \text{ “ } B = R^+ \text{ “ } A$ 
    proof
      show  $B \cup R'^+ \text{ “ } B \subseteq R^+ \text{ “ } A$  unfolding B-def R using mono
      by blast
    next
      show  $R^+ \text{ “ } A \subseteq B \cup R'^+ \text{ “ } B$ 
      proof
        fix x
        assume  $x \in R^+ \text{ “ } A$ 
        then obtain a where  $a: a \in A$  and  $ax: (a,x) \in R^+$  by auto
        from ax a show  $x \in B \cup R'^+ \text{ “ } B$ 
        proof (induct)
          case base
          then show ?case unfolding B-def by auto
        next
          case (step x y)

```

```

from step(3)[OF step(4)] have  $x: x \in B \cup R'^\wedge +$  “  $B$  .
show ?case
proof (cases  $y \in B$ )
  case False note  $y = \text{this}$ 
  show ?thesis
  proof (cases  $x \in A$ )
    case True
    with  $y \text{ step}(2)$  show ?thesis unfolding B-def by auto
  next
  case False
  with  $y \text{ step}(2)$  have  $(x,y) \in R'$  unfolding R'-def by auto
  with  $x$  have  $y \in (R' \cup (R'^\wedge + O R'))$  “  $B$  by blast
  also have  $R' \cup (R'^\wedge + O R') = R'^\wedge +$  by regex
  finally show ?thesis by blast
qed
qed auto
qed
qed
qed
with False show  $R^+$  “  $A = (\text{if } B \subseteq \{\} \text{ then } \{\} \text{ else } B \cup R'^\wedge +$  “  $B)$  by auto
qed
qed

lemma [code-unfold]:  $R^\wedge +$  “  $A = \text{compute-trancl } A \text{ } R$  unfolding compute-trancl-def
by auto
lemma compute-rtrancl[code-unfold]:  $R^\wedge *$  “  $A = A \cup \text{compute-trancl } A \text{ } R$ 
proof –
  have id:  $R^\wedge * = (Id \cup R^\wedge +)$  by regex
  show ?thesis unfolding id compute-trancl-def by auto
qed
lemma [code-unfold]:  $(a,b) \in R^\wedge + \longleftrightarrow b \in \text{compute-trancl } \{a\} \text{ } R$  unfolding
compute-trancl-def by auto
lemma [code-unfold]:  $(a,b) \in R^\wedge * \longleftrightarrow b = a \vee b \in \text{compute-trancl } \{a\} \text{ } R$ 
using compute-rtrancl[of R {a}] by auto

end

```

1 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*: $\text{cop } \text{nil } x = x$

assumes *cop-add*: $\text{cop } (\text{add } a \text{ } b) \text{ } x = \text{cop } a \text{ } (\text{cop } b \text{ } x)$

begin

inductive-set *closure* **for** $r :: 'a \text{ rel}$

where

$[intro]: (x, y) \in r \implies (cop\ a\ x, cop\ a\ y) \in closure\ r$

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

lemma *closure-mono*: $r \subseteq s \implies closure\ r \subseteq closure\ s$ **by** (*auto elim: closure.cases*)

lemma *subset-closure*: $r \subseteq closure\ r$

using *closure.intros* [**where** $a = nil$] **by** (*auto simp: cop-nil*)

definition *closed* $r \longleftrightarrow closure\ r \subseteq r$

lemma *closure-subset*: $closed\ r \implies closure\ r \subseteq r$

by (*auto simp: closed-def*)

lemma *closedI* [*Pure.intro, intro*]: $(\bigwedge x\ y\ a. (x, y) \in r \implies (cop\ a\ x, cop\ a\ y) \in r) \implies closed\ r$

by (*auto simp: closed-def elim: closure.cases*)

lemma *closedD* [*dest*]: $closed\ r \implies (x, y) \in r \implies (cop\ a\ x, cop\ a\ y) \in r$

by (*auto simp: closed-def*)

lemma *closed-closure* [*intro*]: $closed\ (closure\ r)$

using *closure.intros* [**where** $a = add\ a\ b$ **for** $a\ b$]

by (*auto simp: closed-def cop-add elim!: closure.cases*)

lemma *subset-closure-Un*:

$closure\ r \subseteq closure\ (r \cup s)$

$closure\ s \subseteq closure\ (r \cup s)$

by (*auto elim!: closure.cases*)

lemma *closure-Un*: $closure\ (r \cup s) = closure\ r \cup closure\ s$

using *subset-closure-Un* **by** (*auto elim: closure.cases*)

lemma *closure-id* [*simp*]: $closed\ r \implies closure\ r = r$

using *subset-closure* **and** *closure-subset* **by** *blast*

lemma *closed-Un* [*intro*]: $closed\ r \implies closed\ s \implies closed\ (r \cup s)$ **by** *blast*

lemma *closed-Inr* [*intro*]: $closed\ r \implies closed\ s \implies closed\ (r \cap s)$ **by** *blast*

lemma *closed-rtrancl* [*intro*]: $closed\ r \implies closed\ (r^*)$

by (*best intro: rtrancl-into-rtrancl elim: rtrancl.induct*)

lemma *closed-trancl* [*intro*]: $closed\ r \implies closed\ (r^+)$

by (*best intro: trancl-into-trancl elim: trancl.induct*)
lemma *closed-converse* [*intro*]: $\text{closed } r \implies \text{closed } (r^{-1})$ **by** *blast*
lemma *closed-comp* [*intro*]: $\text{closed } r \implies \text{closed } s \implies \text{closed } (r \circ s)$ **by** *blast*
lemma *closed-relpow* [*intro*]: $\text{closed } r \implies \text{closed } (r^{\sim n})$
by (*auto intro: relpow-image [OF closedD]*)
lemma *closed-conversion* [*intro*]: $\text{closed } r \implies \text{closed } (r^{\leftrightarrow*})$
by (*auto simp: conversion-def*)
lemma *closed-relto* [*intro*]: $\text{closed } r \implies \text{closed } s \implies \text{closed } (\text{relto } r \ s)$ **by** *blast*
lemma *closure-diff-subset*: $\text{closure } r - \text{closure } s \subseteq \text{closure } (r - s)$ **by** (*auto elim: closure.cases*)
end
end

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

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 $P \ s \ t$
using *assms* **by** (*induct*) *simp*

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  $P\ s\ t$ 
using assms by (induct) auto

lemmas rstepI = rstep.intros [intro]

lemmas rstepE = rstep.cases [elim]

lemma rstep-ctxt [intro]:  $(s, t) \in \text{rstep } R \implies (C\langle s \rangle, C\langle t \rangle) \in \text{rstep } R$ 
by (force simp flip: ctxt-ctxt-compose)

lemma rstep-rule [intro]:  $(l, r) \in R \implies (l, r) \in \text{rstep } R$ 
using rstep.rstep [where  $C = \square$  and  $\sigma = \text{Var}$  and  $R = R$ ] by simp

lemma rstep-subst [intro]:  $(s, t) \in \text{rstep } R \implies (s \cdot \sigma, t \cdot \sigma) \in \text{rstep } R$ 
by (force simp flip: subst-subst-compose)

lemma rstep-empty [simp]:  $\text{rstep } \{\} = \{\}$ 
by auto

lemma rstep-mono:  $R \subseteq S \implies \text{rstep } R \subseteq \text{rstep } S$ 
by force

lemma rstep-union:  $\text{rstep } (R \cup S) = \text{rstep } R \cup \text{rstep } S$ 
by auto

lemma rstep-converse [simp]:  $\text{rstep } (R^{-1}) = (\text{rstep } R)^{-1}$ 
by auto

interpretation subst: rel-closure  $\lambda\sigma\ t. t \cdot \sigma$  Var  $\lambda x\ y. y \circ_s x$  by (standard) auto
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  $\square (\circ_c)$  by (standard) auto
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:  $\text{rstep } R = \text{ctxt.closure } (\text{subst.closure } R)$ 
by (force elim: ctxt.closure.cases subst.closure.cases)

lemma ctxt-closed-rstep [intro]:  $\text{ctxt.closed } (\text{rstep } R)$ 
by (simp add: rstep-eq-closure ctxt.closed-closure)

lemma ctxt-closed-one:
 $\text{ctxt.closed } r \implies (s, t) \in r \implies (\text{Fun } f\ (ss @ s \# ts), \text{Fun } f\ (ss @ t \# ts)) \in r$ 

```

```

using ctxt.closedD [of r s t More f ss □ ts] by auto

lemma args-steps-imp-steps:
  assumes ctxt.closed r
    and len: length ts = length us
    and  $\forall i < \text{length } us. (ts ! i, us ! i) \in r^*$ 
  shows  $(\text{Fun } f \text{ } ts, \text{Fun } f \text{ } us) \in r^*$ 
proof -
  have  $(\text{Fun } f \text{ } ts, \text{Fun } f \text{ } (\text{take } i \text{ } us @ \text{drop } i \text{ } ts)) \in r^*$  if  $i \leq \text{length } us$  for  $i$ 
    using that
  proof (induct i)
    case (Suc i)
    then have  $(\text{Fun } f \text{ } ts, \text{Fun } f \text{ } (\text{take } i \text{ } us @ \text{drop } i \text{ } ts)) \in r^*$  by simp
    moreover have  $\text{take } i \text{ } us @ \text{drop } i \text{ } ts = \text{take } i \text{ } us @ ts ! \text{length } (\text{take } i \text{ } us) \#$ 
       $\text{drop } (\text{Suc } i) \text{ } ts$ 
    using Suc and len by (auto simp: Cons-nth-drop-Suc min-def)
    moreover have  $\text{take } (\text{Suc } i) \text{ } us @ \text{drop } (\text{Suc } i) \text{ } ts = \text{take } i \text{ } us @ us ! \text{length } (\text{take } i \text{ } us) \#$ 
       $\text{drop } (\text{Suc } i) \text{ } ts$ 
    using Suc and len by (auto simp: min-def take-Suc-conv-app-nth)
    moreover have  $(\text{Fun } f \text{ } (\text{take } i \text{ } us @ ts ! \text{length } (\text{take } i \text{ } us) \# \text{drop } (\text{Suc } i) \text{ } ts),$ 
       $\text{Fun } f \text{ } (\text{take } i \text{ } us @ us ! \text{length } (\text{take } i \text{ } us) \# \text{drop } (\text{Suc } i) \text{ } ts)) \in r^*$ 
    using assms and Suc by (intro ctxt-closed-one) auto
    ultimately show ?case by simp
  qed simp
  note this [of length us] and len
  then show ?thesis by simp
qed

end

```

3 Term Rewrite Systems

```

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

```

3.1 Variables of Rules

```

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

lemma finite-vars-rule:
  finite (vars-rule r)
  by (auto simp: vars-rule-def)

```

3.2 Variables of TRSs

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

lemma vars-trs-union : $\text{vars-trs } (R \cup S) = \text{vars-trs } R \cup \text{vars-trs } S$
unfolding vars-trs-def **by** *auto*

lemma $\text{finite-trs-has-finite-vars}$:
assumes $\text{finite } R$ **shows** $\text{finite } (\text{vars-trs } R)$
using *assms* **unfolding** vars-trs-def vars-rule-def $[\text{abs-def}]$ **by** *simp*

lemmas $\text{vars-defs} = \text{vars-trs-def } \text{vars-rule-def}$

3.3 Function Symbols of Rules

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

The same including arities.

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

3.4 Function Symbols of TRSs

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

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

lemma $\text{funs-rule-funas-rule}$: $\text{funs-rule } rl = \text{fst } ' \text{funas-rule } rl$
using $\text{funs-term-funas-term}$ **unfolding** funs-rule-def funas-rule-def image-Un **by** *metis*

lemma $\text{funs-trs-funas-trs}$: $\text{funs-trs } R = \text{fst } ' \text{funas-trs } R$
unfolding funs-trs-def funas-trs-def image-UN **using** $\text{funs-rule-funas-rule}$ **by** *metis*

lemma finite-funas-rule : $\text{finite } (\text{funas-rule } lr)$
unfolding funas-rule-def
using finite-funas-term **by** *auto*

lemma finite-funas-trs :
assumes $\text{finite } R$
shows $\text{finite } (\text{funas-trs } R)$
unfolding funas-trs-def
using *assms* finite-funas-rule **by** *auto*

lemma $\text{funas-empty}[simp]$: $\text{funas-trs } \{\} = \{\}$ **unfolding** funas-trs-def **by** *simp*

lemma *funas-trs-union*[simp]: *funas-trs* ($R \cup S$) = *funas-trs* $R \cup$ *funas-trs* S
unfolding *funas-trs-def* **by** *blast*

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* $P - (\text{funas-trs } R \cup \text{funas-args-trs } P)$

lemmas *funs-defs* = *funs-trs-def funs-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* $\Rightarrow ('f \times \text{nat}) \Rightarrow \text{bool}$
where
defined $R \text{ fn}$ $\longleftrightarrow (\exists l \text{ } r. (l, r) \in R \wedge \text{root } l = \text{Some } \text{fn})$

lemma *defined-funas-trs*: **assumes** d : *defined* $R \text{ fn}$ **shows** $\text{fn} \in \text{funas-trs } R$
proof –

from d [*unfolded defined-def*] **obtain** $l \text{ } r$
where $(l, r) \in R$ **and** $\text{root } l = \text{Some } \text{fn}$ **by** *auto*
then show *?thesis*
unfolding *funas-trs-def funas-rule-def [abs-def]* **by** (*cases l*) *force+*
qed

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

assumes *ctxt.closed* *R* **and** $s \triangleright t$ **and** $(t, u) \in R$ **shows** $\exists t. (s, t) \in R \wedge t \triangleright u$
proof –
from $\langle s \triangleright t \rangle$ **obtain** *C* **where** $C \neq \square$ **and** $C\langle t \rangle = s$ **by** *auto*
from $\langle \text{ctxt.closed } R \rangle$ **and** $\langle (t, u) \in R \rangle$
have $R\text{CtCu}: (C\langle t \rangle, C\langle u \rangle) \in R$ **by** (rule *ctxt.closedD*)
from $\langle C \neq \square \rangle$ **have** $C\langle u \rangle \triangleright u$ **by** *auto*
from $R\text{CtCu}$ **have** $(s, C\langle u \rangle) \in R$ **unfolding** $\langle C\langle t \rangle = s \rangle$.
from *this* **and** $\langle C\langle u \rangle \triangleright u \rangle$ **show** *?thesis* **by** *auto*
qed

lemma *ctxt-closed-imp-qc-supt*: $\text{ctxt.closed } R \implies \{\triangleright\} \circ R \subseteq R \circ (R \cup \{\triangleright\})^*$
by *blast*

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* $(R \cup \{\triangleright\})$
proof –
from $\langle \text{ctxt.closed } R \rangle$ **have** *quasi-commute* $R \{\triangleright\}$
unfolding *quasi-commute-def* **by** (rule *ctxt-closed-imp-qc-supt*)
have *SN* $\{\triangleright\}$ **by** (rule *SN-supt*)
from $\langle \text{SN } R \rangle$ **and** $\langle \text{SN } \{\triangleright\} \rangle$ **and** $\langle \text{quasi-commute } R \{\triangleright\} \rangle$
show *?thesis* **by** (rule *quasi-commute-imp-SN*)
qed

lemma *stable-loop-imp-not-SN*:
assumes *stable*: *subst.closed* *r* **and** *steps*: $(s, s \cdot \sigma) \in r^+$
shows $\neg \text{SN-on } r \{s\}$
proof –
let $?f = \lambda i. s \cdot (\text{power.power Var } (\circ_s) \sigma i)$
have *main*: $\bigwedge i. (?f i, ?f (\text{Suc } i)) \in r^+$
proof –
fix *i*
show $(?f i, ?f (\text{Suc } i)) \in r^+$
proof (*induct i*)
case (*Suc i*)
from *Suc* *subst.closed-tranc1*[*OF stable*] **have** *step*: $(?f i \cdot \sigma, ?f (\text{Suc } i) \cdot \sigma) \in r^+$ **by** *auto*
let $?sg = \text{power.power Var } (\circ_s) \sigma i$
let $?sgs = \text{power.power Var } (\circ_s) \sigma (\text{Suc } i)$
have *idi*: $?sg \circ_s \sigma = \sigma \circ_s ?sg$ **by** (rule *subst-monoid-mult.power-commutes*)
have *idsi*: $?sgs \circ_s \sigma = \sigma \circ_s ?sgs$ **by** (rule *subst-monoid-mult.power-commutes*)
have $?f i \cdot \sigma = s \cdot ?sg \circ_s \sigma$ **by** *simp*
also have $\dots = ?f (\text{Suc } i)$ **unfolding** *idi* **by** *simp*
finally have *one*: $?f i \cdot \sigma = ?f (\text{Suc } i)$.
have $?f (\text{Suc } i) \cdot \sigma = s \cdot ?sgs \circ_s \sigma$ **by** *simp*
also have $\dots = ?f (\text{Suc } (\text{Suc } i))$ **unfolding** *idsi* **by** *simp*
finally have *two*: $?f (\text{Suc } i) \cdot \sigma = ?f (\text{Suc } (\text{Suc } i))$ **by** *simp*

```

    show ?case using one two step by simp
  qed (auto simp: steps)
qed
then have  $\neg$  SN-on  $(r^+)$   $\{?f\ 0\}$  unfolding SN-on-def by best
then show ?thesis using SN-on-trancl by force
qed

lemma subst-closed-supteq: subst.closed  $\{\supseteq\}$  by blast

lemma subst-closed-supt: subst.closed  $\{\triangleright\}$  by blast

lemma rstep-relcomp-idemp1 [simp]:
  rstep (rstep R O rstep S) = rstep R O rstep S
proof -
  { fix s t
    assume  $(s, t) \in \text{rstep } (rstep R \ O \ rstep S)$ 
    then have  $(s, t) \in \text{rstep } R \ O \ rstep S$ 
      by (induct) blast+ }
  then show ?thesis by auto
qed

lemma rstep-relcomp-idemp2 [simp]:
  rstep (rstep R O rstep S O rstep T) = rstep R O rstep S O rstep T
proof -
  { fix s t
    assume  $(s, t) \in \text{rstep } (rstep R \ O \ rstep S \ O \ rstep T)$ 
    then have  $(s, t) \in \text{rstep } R \ O \ rstep S \ O \ rstep T$ 
      by (induct) blast+ }
  then show ?thesis by auto
qed

lemma ctxt-closed-rsteps [intro]: ctxt.closed  $((\text{rstep } R)^*)$  by blast

lemma subset-rstep:  $R \subseteq \text{rstep } R$  by auto

lemma subst-closure-rstep-subset: subst.closure (rstep R)  $\subseteq \text{rstep } R$ 
by (auto elim: subst.closure.cases)

lemma subst-closed-rstep [intro]: subst.closed (rstep R) by blast

lemma subst-closed-rsteps: subst.closed  $((\text{rstep } R)^*)$  by blast

lemma ctxt-closed-supt-subset: ctxt.closed  $R \implies \{\triangleright\} \ O \ R \subseteq R \ O \ \{\triangleright\}$  by blast

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

lemma supseq-rsteps-subset:
   $\{\supseteq\} \ O \ (\text{rstep } R)^* \subseteq (\text{rstep } R)^* \ O \ \{\supseteq\}$  (is  $?S \subseteq ?T$ )
using supt-rsteps-subset [of R] by (auto simp: supt-supteq-set-conv)

```

lemma *quasi-commute-rsteps-supt*:

quasi-commute $((rstep\ R)^*) \{\triangleright\}$

unfolding *quasi-commute-def* **using** *supt-rsteps-subset* [of *R*] **by** *auto*

lemma *rstep-UN*:

$rstep\ (\bigcup_{i \in A} R\ i) = (\bigcup_{i \in A} rstep\ (R\ i))$

by (*force*)

definition

$rstep\text{-}r\text{-}p\text{-}s :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ rule \Rightarrow pos \Rightarrow ('f, 'v)\ subst \Rightarrow ('f, 'v)\ trs$

where

$rstep\text{-}r\text{-}p\text{-}s\ R\ r\ p\ \sigma = \{(s, t) \mid$

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

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

$rstep\text{-}r\text{-}p\text{-}s\ R\ r\ p\ \sigma = \{(s, t) \mid$

$p \in \text{poss } s \wedge r \in R \wedge s \mid - p = \text{fst } r \cdot \sigma \wedge t = \text{replace-at } s\ p\ (\text{snd } r \cdot \sigma)\} \text{ (is } ?l = ?r)$

proof –

{ **fix** *s t*

have $((s, t) \in ?l) = ((s, t) \in ?r)$

unfolding *rstep-r-p-s-def* *Let-def*

using *ctxt-supt-id* [of *p s*] **and** *subt-at-ctxt-of-pos-term* [of *p s fst r · σ*] **by**

auto }

then show *?thesis* **by** *auto*

qed

lemma *parallel-steps*:

fixes $p_1 :: pos$

assumes $(s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R_1\ (l_1, r_1)\ p_1\ \sigma_1$

and $(s, u) \in rstep\text{-}r\text{-}p\text{-}s\ R_2\ (l_2, r_2)\ p_2\ \sigma_2$

and *par*: $p_1 \perp p_2$

shows $(t, (\text{ctxt-of-pos-term } p_1\ u)\langle r_1 \cdot \sigma_1 \rangle) \in rstep\text{-}r\text{-}p\text{-}s\ R_2\ (l_2, r_2)\ p_2\ \sigma_2 \wedge$

$(u, (\text{ctxt-of-pos-term } p_1\ u)\langle r_1 \cdot \sigma_1 \rangle) \in rstep\text{-}r\text{-}p\text{-}s\ R_1\ (l_1, r_1)\ p_1\ \sigma_1$

proof –

have *p1*: $p_1 \in \text{poss } s$ **and** *lr1*: $(l_1, r_1) \in R_1$ **and** *σ1*: $s \mid - p_1 = l_1 \cdot \sigma_1$

and *t*: $t = \text{replace-at } s\ p_1\ (r_1 \cdot \sigma_1)$

and *p2*: $p_2 \in \text{poss } s$ **and** *lr2*: $(l_2, r_2) \in R_2$ **and** *σ2*: $s \mid - p_2 = l_2 \cdot \sigma_2$

and *u*: $u = \text{replace-at } s\ p_2\ (r_2 \cdot \sigma_2)$ **using** *assms* **by** (*auto simp: rstep-r-p-s-def'*)

have $\text{replace-at } t\ p_2\ (r_2 \cdot \sigma_2) = \text{replace-at } u\ p_1\ (r_1 \cdot \sigma_1)$

using *t* **and** *u* **and** *parallel-replace-at* [*OF* $\langle p_1 \perp p_2 \rangle\ p1\ p2$] **by** *auto*

moreover

have $(t, (\text{ctxt-of-pos-term } p_2\ t)\langle r_2 \cdot \sigma_2 \rangle) \in rstep\text{-}r\text{-}p\text{-}s\ R_2\ (l_2, r_2)\ p_2\ \sigma_2$

proof –

have *t* $\mid - p_2 = l_2 \cdot \sigma_2$ **using** *σ2* **and** *parallel-replace-at-subt-at* [*OF par p1 p2*]

and *t* **by** *auto*

moreover have $p_2 \in \text{poss } t$ using *parallel-poss-replace-at* [*OF par p1*] and t
 and p_2 by *auto*
 ultimately show *?thesis* using *lr2* and *ctxt-supt-id* [*of p2 t*] by (*simp add:*
rstep-r-p-s-def)
 qed
 moreover
 have $(u, (\text{ctxt-of-pos-term } p_1 \ u) \langle r_1 \cdot \sigma_1 \rangle) \in \text{rstep-r-p-s } R_1 \ (l_1, r_1) \ p_1 \ \sigma_1$
 proof -
 have $\text{par}' : p_2 \perp p_1$ using *parallel-pos-sym* [*OF par*] .
 have $u \vdash p_1 = l_1 \cdot \sigma_1$ using σ_1 and *parallel-replace-at-subt-at* [*OF par' p2*
p1] and u by *auto*
 moreover have $p_1 \in \text{poss } u$ using *parallel-poss-replace-at* [*OF par' p2*] and
 u and p_1 by *auto*
 ultimately show *?thesis* using *lr1* and *ctxt-supt-id* [*of p1 u*] by (*simp add:*
rstep-r-p-s-def)
 qed
 ultimately show *?thesis* by *auto*
 qed

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

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

assume $(s, t) \in \text{rstep } R$
 then obtain $C \ \sigma \ l \ r$ where $s : s = C \langle l \cdot \sigma \rangle$ and $t : t = C \langle r \cdot \sigma \rangle$ and $(l, r) \in$
 R by *auto*
 let $?p = \text{hole-pos } C$
 let $?C = \text{ctxt-of-pos-term } ?p \ s$
 have $C : \text{ctxt-of-pos-term } ?p \ s = C$ unfolding s by (*induct C*) *simp-all*
 have $?p \in \text{poss } s$ unfolding s by *simp*
 moreover have $(l, r) \in R$ by *fact*
 moreover have $?C \langle l \cdot \sigma \rangle = s$ unfolding C by (*simp add: s*)
 moreover have $?C \langle r \cdot \sigma \rangle = t$ unfolding C by (*simp add: t*)
 ultimately show *?rhs* unfolding *rstep-r-p-s-def* *Let-def* by *auto*
 next
 assume *?rhs*
 then obtain $l \ r \ p \ \sigma$ where $p \in \text{poss } s$
 and $(l, r) \in R$
 and $s[\text{symmetric}] : (\text{ctxt-of-pos-term } p \ s) \langle l \cdot \sigma \rangle = s$
 and $t[\text{symmetric}] : (\text{ctxt-of-pos-term } p \ s) \langle r \cdot \sigma \rangle = t$
 unfolding *rstep-r-p-s-def* *Let-def* by *auto*
 then show *?lhs* by *auto*
 qed

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

assumes $(s, t) \in \text{rstep-r-p-s } R \ r \ p \ \sigma$
 shows $(s, t) \in \text{rstep } R$
 using *assms* by (*cases r*) (*auto simp: rstep-iff-rstep-r-p-s*)

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 $nrrstep\text{-}def'$:

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

proof

show $?lhs \subseteq ?rhs$

proof (*rule subrelI*)

fix $s \ t$ **assume** $(s, t) \in nrrstep \ R$

then obtain $l \ r \ i \ ps \ \sigma$ **where** $step: (s, t) \in rstep\text{-}r\text{-}p\text{-}s \ R \ (l, r) \ (i \# ps) \ \sigma$

unfolding $nrrstep\text{-}def$ **by** *best*

let $?C = ctxt\text{-}of\text{-}pos\text{-}term \ (i \# ps) \ s$

from $step$ **have** $i \# ps \in poss \ s$ **and** $(l, r) \in R$ **and** $s = ?C\langle l \cdot \sigma \rangle$ **and** $t = ?C\langle r \cdot \sigma \rangle$

unfolding $rstep\text{-}r\text{-}p\text{-}s\text{-}def$ *Let-def* **by** *auto*

moreover from $\langle i \# ps \in poss \ s \rangle$ **have** $?C \neq \square$ **by** (*induct s*) *auto*

ultimately show $(s, t) \in ?rhs$ **by** *auto*

qed

next

show $?rhs \subseteq ?lhs$

proof (*rule subrelI*)

fix $s \ t$ **assume** $(s, t) \in ?rhs$

then obtain $l \ r \ C \ \sigma$ **where** $in\text{-}R: (l, r) \in R$ **and** $C \neq \square$

and $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$ **by** *auto*

from $\langle C \neq \square \rangle$ **obtain** $i \ p$ **where** $ip: hole\text{-}pos \ C = i \# p$ **by** (*induct C*) *auto*

have $i \# p \in poss \ s$ **using** $hole\text{-}pos\text{-}poss[of \ C]$ **unfolding** $s \ ip$ **by** *simp*

then have $C: C = ctxt\text{-}of\text{-}pos\text{-}term \ (i \# p) \ s$

unfolding $s \ ip[symmetric]$ **by** *simp*

from $\langle i \# p \in poss \ s \rangle$ $in\text{-}R \ s \ t$ **have** $(s, t) \in rstep\text{-}r\text{-}p\text{-}s \ R \ (l, r) \ (i \# p) \ \sigma$

unfolding $rstep\text{-}r\text{-}p\text{-}s\text{-}def \ C$ **by** *simp*

then show $(s, t) \in nrrstep \ R$ **unfolding** $nrrstep\text{-}def$ **by** *best*

qed

qed

lemma $nrrstepI: (l, r) \in R \Longrightarrow s = C\langle l \cdot \sigma \rangle \Longrightarrow t = C\langle r \cdot \sigma \rangle \Longrightarrow C \neq \square \Longrightarrow (s, t) \in nrrstep \ R$ **unfolding** $nrrstep\text{-}def'$ **by** *auto*

lemma $nrrstep\text{-}union: nrrstep \ (R \cup S) = nrrstep \ R \cup nrrstep \ S$

unfolding $nrrstep\text{-}def'$ **by** *blast*

lemma $nrrstep\text{-}empty[simp]: nrrstep \ \{\} = \{\}$ **unfolding** $nrrstep\text{-}def'$ **by** *blast*

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)

by (auto simp: rrstep-def rstep-r-p-s-def)

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

by (force simp: rrstep-def' intro: exI [of - Var])

lemma *rrstep-union*: $rrstep\ (R \cup S) = rrstep\ R \cup rrstep\ S$ **unfolding** *rrstep-def'*
by blast

lemma *rrstep-empty*[simp]: $rrstep\ \{\} = \{\}$
unfolding *rrstep-def'* **by** auto

lemma *subst-closed-rrstep*: *subst.closed* (*rrstep* *R*)

unfolding *subst.closed-def*

proof

fix *ss ts*

assume $(ss, ts) \in \text{subst.closure}\ (rrstep\ R)$

then show $(ss, ts) \in rrstep\ R$

proof

fix *s t* σ

assume *ss*: $ss = s \cdot \sigma$ and *ts*: $ts = t \cdot \sigma$ and *step*: $(s, t) \in rrstep\ R$

from *step* obtain *l r* δ where *lr*: $(l, r) \in R$ and *s*: $s = l \cdot \delta$ and *t*: $t = r \cdot \delta$

unfolding *rrstep-def'* **by** auto

obtain *sig* where $sig = \delta \circ_s \sigma$ **by** auto

with *ss s ts t* have $ss = l \cdot sig$ and $ts = r \cdot sig$ **by** simp+

with *lr* show $(ss, ts) \in rrstep\ R$ **unfolding** *rrstep-def'* **by** (auto simp: Let-def)

qed

qed

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

proof

show $rstep\ R \subseteq rrstep\ R \cup nrrstep\ R$

proof (rule subrelI)

fix *s t* assume $(s, t) \in rstep\ R$

then obtain *l r p* σ where *rstep-rps*: $(s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ (l, r)\ p\ \sigma$

by (auto simp: rstep-iff-rstep-r-p-s)

then show $(s, t) \in rrstep\ R \cup nrrstep\ R$ **unfolding** *rrstep-def* *nrrstep-def* **by** (cases *p*) auto

qed

next

show $rrstep\ R \cup nrrstep\ R \subseteq rstep\ R$

proof (rule subrelI)

fix *s t* assume $(s, t) \in rrstep\ R \cup nrrstep\ R$

then show $(s, t) \in \text{rstep } R$ **by** $(\text{auto simp: rstep-def nrrstep-def rstep-iff-rstep-r-p-s})$
qed
qed

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

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

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

proof –

from *nrrstep* **obtain** C **where** $C : C = \text{ctxt-of-pos-term } (i \# ps) \ (\text{Fun } f \text{ ss})$

and *pos*: $(i \# ps) \in \text{poss } (\text{Fun } f \text{ ss})$

and *Rlr*: $(l, r) \in R$

and *Fun*: $C \langle l \cdot \sigma \rangle = \text{Fun } f \text{ ss}$

and *t*: $C \langle r \cdot \sigma \rangle = t$ **unfolding** *rstep-r-p-s-def Let-def* **by** *auto*

then obtain D **where** $C' : C = \text{More } f \ (\text{take } i \text{ ss}) \ D \ (\text{drop } (\text{Suc } i) \text{ ss})$ **by** *auto*

then have $C \text{Fun}$: $C \langle l \cdot \sigma \rangle = \text{Fun } f \ (\text{take } i \text{ ss} @ (D \langle l \cdot \sigma \rangle) \# \text{drop } (\text{Suc } i) \text{ ss})$ **by** *auto*

from *pos* **have** *len*: $i < \text{length ss}$ **by** *auto*

from *len*

have $(\text{take } i \text{ ss} @ (D \langle l \cdot \sigma \rangle) \# \text{drop } (\text{Suc } i) \text{ ss})!i = D \langle l \cdot \sigma \rangle$ **by** $(\text{auto simp: nth-append})$

with $C \text{Fun } C \text{Fun}$ **have** *ssi*: $\text{ss}!i = D \langle l \cdot \sigma \rangle$ **by** *auto*

from $C' \ t$ **have** *t'*: $t = \text{Fun } f \ (\text{take } i \text{ ss} @ (D \langle r \cdot \sigma \rangle) \# \text{drop } (\text{Suc } i) \text{ ss})$ **by** *auto*

from *len*

have $(\text{take } i \text{ ss} @ (D \langle r \cdot \sigma \rangle) \# \text{drop } (\text{Suc } i) \text{ ss})!i = D \langle r \cdot \sigma \rangle$ **by** $(\text{auto simp: nth-append})$

with *t'* **have** $t[-i] = (D \langle r \cdot \sigma \rangle)[-i]$ **by** *auto*

then have *subt-at*: $t[-i] = D \langle r \cdot \sigma \rangle$ **by** *simp*

from $C \ C'$ **have** $D = \text{ctxt-of-pos-term } ps \ (\text{ss}!i)$ **by** *auto*

with *pos* *Rlr* *ssi*[*symmetric*] *subt-at*[*symmetric*]

show *?thesis* **unfolding** *rstep-r-p-s-def Let-def* **by** *auto*

qed

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

by $(\text{rule ctxt.closure-id})$ *blast*

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

by $(\text{rule subst.closure-id})$ *blast*

lemma *supt-rstep-subset*:

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

proof (rule subrelI)

fix $s \ t$ **assume** $(s, t) \in \{\triangleright\} \ O \ \text{rstep } R$ **then show** $(s, t) \in \text{rstep } R \ O \ \{\triangleright\}$

proof $(\text{induct } s)$

case $(\text{Var } x)$

then have $\exists u. \text{Var } x \triangleright u \wedge (u, t) \in \text{rstep } R$ **by** *auto*

then obtain u **where** $\text{Var } x \triangleright u$ **and** $(u, t) \in \text{rstep } R$ **by** *auto*

from $\langle \text{Var } x \triangleright u \rangle$ **show** *?case* **by** $(\text{cases rule: supt.cases})$

next

case $(\text{Fun } f \text{ ss})$

then obtain u **where** $\text{Fun } f \text{ ss} \triangleright u$ **and** $(u, t) \in \text{rstep } R$ **by** *auto*

from $\langle \text{Fun } f \text{ ss} \triangleright u \rangle$ **obtain** C

```

    where  $C \neq \square$  and  $C\langle u \rangle = \text{Fun } f \text{ ss}$  by auto
    from  $\langle C \neq \square \rangle$  have  $C\langle t \rangle \triangleright t$  by (rule nectxt-imp-supt-ctxt)
    from  $\langle (u, t) \in \text{rstep } R \rangle$  have  $(C\langle u \rangle, C\langle t \rangle) \in \text{rstep } R$  ..
    then have  $(\text{Fun } f \text{ ss}, C\langle t \rangle) \in \text{rstep } R$  unfolding  $\langle C\langle u \rangle = \text{Fun } f \text{ ss} \rangle$  .
    with  $\langle C\langle t \rangle \triangleright t \rangle$  show ?case by auto
  qed
qed

lemma ne-rstep-seq-imp-list-of-terms:
  assumes  $(s, t) \in (\text{rstep } R)^+$ 
  shows  $\exists ts. \text{length } ts > 1 \wedge ts!0 = s \wedge ts!(\text{length } ts - 1) = t \wedge$ 
     $(\forall i < \text{length } ts - 1. (ts!i, ts!(\text{Suc } i)) \in (\text{rstep } R))$  (is  $\exists ts. - \wedge - \wedge - \wedge ?P \text{ ts}$ )
  using assms
proof (induct rule: trancl.induct)
  case (r-into-trancl x y)
  let ?ts = [x, y]
  have  $\text{length } ?ts > 1$  by simp
  moreover have  $?ts!0 = x$  by simp
  moreover have  $?ts!(\text{length } ?ts - 1) = y$  by simp
  moreover from r-into-rtrancl r-into-trancl have  $?P \text{ ?ts}$  by auto
  ultimately show ?case by fast
next
  case (trancl-into-trancl x y z)
  then obtain ts where  $\text{length } ts > 1$  and  $ts!0 = x$ 
    and last1:  $ts!(\text{length } ts - 1) = y$  and  $?P \text{ ts}$  by auto
  let ?ts = ts@[z]
  have len:  $\text{length } ?ts = \text{length } ts + 1$  by simp
  from  $\langle \text{length } ts > 1 \rangle$  have  $\text{length } ?ts > 1$  by auto
  moreover with  $\langle ts!0 = x \rangle$  have  $?ts!0 = x$  by (induct ts) auto
  moreover have last2:  $?ts!(\text{length } ?ts - 1) = z$  by simp
  moreover have  $?P \text{ ?ts}$ 
  proof (intro allI impI)
    fix i assume A:  $i < \text{length } ?ts - 1$ 
    show  $(?ts!i, ?ts!(\text{Suc } i)) \in \text{rstep } R$ 
    proof (cases  $i < \text{length } ts - 1$ )
      case True with  $\langle ?P \text{ ts} \rangle$  A show ?thesis unfolding len unfolding nth-append
    by auto
    next
      case False with A have  $i = \text{length } ts - 1$  by simp
      with last1  $\langle \text{length } ts > 1 \rangle$  have  $?ts!i = y$  unfolding nth-append by auto
      have  $\text{Suc } i = \text{length } ?ts - 1$  using  $\langle i = \text{length } ts - 1 \rangle$  using  $\langle \text{length } ts > 1 \rangle$  by auto
      with last2 have  $?ts!(\text{Suc } i) = z$  by auto
      from  $\langle (y, z) \in \text{rstep } R \rangle$  show ?thesis unfolding  $\langle ?ts!(\text{Suc } i) = z \rangle \langle ?ts!i = y \rangle$ 
    .
  qed
  qed
  ultimately show ?case by fast
qed

```

locale *E-compatible* =
 fixes $R :: ('f, 'v)trs$ and $E :: ('f, 'v)trs$
 assumes $E: E \circ R = R \text{ Id} \subseteq E$
begin

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

lemma *ctxt-closed-R-imp-supt-restrict-SN-E-distr*:
 assumes *ctxt.closed* R
 and $(s, t) \in (\text{restrict-SN } (E \circ \{\triangleright\}) \ R)$
 and $(t, u) \in \text{restrict-SN } R \ R$
 shows $(\exists t. (s, t) \in \text{restrict-SN } R \ R \wedge (t, u) \in \text{restrict-SN } (E \circ \{\triangleright\}) \ R) \text{ (is } \exists t. \neg \wedge (t, u) \in ?snSub)$
proof –
 from $\langle (s, t) \in ?snSub \rangle$ **obtain** v **where** *SN-on* $R \ \{s\}$ **and** $ac: (s, v) \in E$ **and** $v \triangleright t$ **unfolding** *restrict-SN-def* *supt-def* **by** *auto*
 from $\langle v \triangleright t \rangle$ **obtain** C **where** $C \neq \text{Hole}$ **and** $v: C \langle t \rangle = v$ **by** *best*
 from $\langle (t, u) \in \text{restrict-SN } R \ R \rangle$ **have** $(t, u) \in R$ **unfolding** *restrict-SN-def* **by** *auto*
 from $\langle \text{ctxt.closed } R \rangle$ **and** *this* **have** $R \text{CtCu}: (C \langle t \rangle, C \langle u \rangle) \in R$ **by** (*rule* *ctxt.closedD*)
 with $v \ ac$ **have** $(s, C \langle u \rangle) \in E \circ R$ **by** *auto*
 then **have** $sCu: (s, C \langle u \rangle) \in R$ **using** E **by** *simp*
 with $\langle \text{SN-on } R \ \{s\} \rangle$ **have** *one*: *SN-on* $R \ \{C \langle u \rangle\}$
 using *step-preserves-SN-on*[*of* $s \ C \langle u \rangle \ R$] **by** *blast*
 from $\langle C \neq \square \rangle$ **have** $C \langle u \rangle \triangleright u$ **by** *auto*
 with *one* E **have** $(C \langle u \rangle, u) \in ?snSub$ **unfolding** *restrict-SN-def* *supt-def* **by** *auto*
 from sCu **and** $\langle \text{SN-on } R \ \{s\} \rangle$ **and** $\langle (C \langle u \rangle, u) \in ?snSub \rangle$ **show** *?thesis* **unfolding** *restrict-SN-def* **by** *auto*
qed

lemma *ctxt-closed-R-imp-restrict-SN-qc-E-supt*:
 assumes *ctxt*: *ctxt.closed* R
 shows *quasi-commute* $(\text{restrict-SN } R \ R) \ (\text{restrict-SN } (E \circ \{\triangleright\} \circ E) \ R)$ (is *quasi-commute* $?r \ ?s$)
proof –
 have $?s \circ ?r \subseteq ?r \circ (?r \cup ?s)^*$
proof (*rule* *subrelI*)
 fix $x \ y$
 assume $(x, y) \in ?s \circ ?r$
 from *this* **obtain** z **where** $(x, z) \in ?s$ **and** $(z, y) \in ?r$ **by** *best*
 then **obtain** $v \ w$ **where** $ac: (x, v) \in E$ **and** $vw: v \triangleright w$ **and** $wz: (w, z) \in E$ **and** $zy: (z, y) \in R$ **and** *SN-on* $R \ \{x\}$ **unfolding** *restrict-SN-def* *supt-def*
 using $E(2)$ **by** *auto*
 from $wz \ zy$ **have** $(w, y) \in E \circ R$ **by** *auto*
 with E **have** $wy: (w, y) \in R$ **by** *auto*
 from *ctxt-closed-R-imp-supt-R-distr*[*OF* *ctxt* $vw \ wy$] **obtain** w **where** $(v, w) \in$

R and $w \triangleright y$ using *ctxt-closed-R-imp-supt-R-distr*[where $R = R$ and $s = v$ and $t = z$ and $u = y$] by *auto*
 with $ac\ E$ have $(x, w) \in R$ and $w \triangleright y$ by *auto*
 from *this* and $\langle SN\text{-on}\ R\ \{x\} \rangle$ have $SN\text{-on}\ R\ \{w\}$ using *step-preserves-SN-on*
 unfolding *supt-supteq-conv* by *auto*
 with $\langle w \triangleright y \rangle\ E$ have $(w, y) \in ?s$ unfolding *restrict-SN-def* *supt-def* by *force*
 with $\langle (x, w) \in R \rangle\ \langle SN\text{-on}\ R\ \{x\} \rangle$ show $(x, y) \in ?r\ O\ (?r \cup ?s)^*$ unfolding
restrict-SN-def by *auto*
 qed
 then show *?thesis* unfolding *quasi-commute-def* .
 qed

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

assumes *ctxt.closed* R
 and SN : $SN\ (E\ O\ \{\triangleright\}\ O\ E)$
 shows $SN\ restrict\text{-}SN\text{-supt}\text{-}E$
proof –
 let $?r = restrict\text{-}SN\ R\ R$
 let $?s = restrict\text{-}SN\ (E\ O\ \{\triangleright\}\ O\ E)\ R$
 from $\langle ctxt.closed\ R \rangle$ have *quasi-commute* $?r\ ?s$
 by (rule *ctxt-closed-R-imp-restrict-SN-qc-E-supt*)
 from SN have $SN\ ?s$ by (rule *SN-subset*, *auto simp: restrict-SN-def*)
 have $SN\ ?r$ by (rule *SN-restrict-SN-idemp*)
 from $\langle SN\ ?r \rangle$ and $\langle SN\ ?s \rangle$ and $\langle quasi\text{-}commute\ ?r\ ?s \rangle$
 show *?thesis* unfolding *restrict-SN-supt-E-def* by (rule *quasi-commute-imp-SN*)
 qed
 end

lemma *E-compatible-Id*: $E\text{-compatible}\ R\ Id$

by *standard auto*

definition *restrict-SN-supt* :: $(f, 'v)\ trs \Rightarrow (f, 'v)\ trs$ **where**

$restrict\text{-}SN\text{-supt}\ R = restrict\text{-}SN\ R\ R \cup restrict\text{-}SN\ \{\triangleright\}\ R$

lemma *ctxt-closed-SN-on-subt*:

assumes *ctxt.closed* R and $SN\text{-on}\ R\ \{s\}$ and $s \supseteq t$
 shows $SN\text{-on}\ R\ \{t\}$
proof (rule *ccontr*)
 assume $\neg SN\text{-on}\ R\ \{t\}$
 then obtain A where $A\ 0 = t$ and $\forall i. (A\ i, A\ (Suc\ i)) \in R$
 unfolding *SN-on-def* by *best*
 from $\langle s \supseteq t \rangle$ obtain C where $s = C\langle t \rangle$ by *auto*
 let $?B = \lambda i. C\langle A\ i \rangle$
 have $\forall i. (?B\ i, ?B\ (Suc\ i)) \in R$
proof
 fix i
 from $\langle \forall i. (A\ i, A\ (Suc\ i)) \in R \rangle$ have $(?B\ i, ?B\ (Suc\ i)) \in ctxt.closure(R)$ by
fast
 then show $(?B\ i, ?B\ (Suc\ i)) \in R$ using $\langle ctxt.closed\ R \rangle$ by *auto*

qed
with $\langle A \ 0 = t \rangle$ **have** $?B \ 0 = s \wedge (\forall i. (?B \ i, ?B(Suc \ i)) \in R)$ **unfolding** $\langle s = C\langle t \rangle \rangle$ **by** *auto*
then **have** $\neg SN\text{-on } R \ \{s\}$ **unfolding** *SN-on-def* **by** *auto*
with *assms* **show** *False* **by** *simp*
qed

lemma *ctxt-closed-R-imp-supt-restrict-SN-distr*:
assumes *R: ctxt.closed R*
and *st: (s,t) ∈ (restrict-SN {▷} R)*
and *tu: (t,u) ∈ restrict-SN R R*
shows $(\exists t. (s,t) \in \text{restrict-SN } R \ R \wedge (t,u) \in \text{restrict-SN } \{\triangleright\} \ R) \text{ (is } \exists t. - \wedge (t,u) \in ?snSub)$
using *E-compatible.ctxt-closed-R-imp-supt-restrict-SN-E-distr[OF E-compatible-Id R - tu, of s]*
st **by** *auto*

lemma *ctxt-closed-R-imp-restrict-SN-qc-supt*:
assumes *ctxt.closed R*
shows *quasi-commute (restrict-SN R R) (restrict-SN supt R) (is quasi-commute ?r ?s)*
using *E-compatible.ctxt-closed-R-imp-restrict-SN-qc-E-supt[OF E-compatible-Id assms]* **by** *auto*

lemma *ctxt-closed-imp-SN-restrict-SN-supt*:
assumes *ctxt.closed R*
shows *SN (restrict-SN-supt R)*
using *E-compatible.ctxt-closed-imp-SN-restrict-SN-E-supt[OF E-compatible-Id assms]*
unfolding *E-compatible.restrict-SN-supt-E-def[OF E-compatible-Id] restrict-SN-supt-def*
using *SN-supt* **by** *auto*

lemma *SN-restrict-SN-supt-rstep*:
shows *SN (restrict-SN-supt (rstep R))*
proof –
have *ctxt.closed (rstep R)* **by** *(rule ctxt-closed-rstep)*
then **show** *?thesis* **by** *(rule ctxt-closed-imp-SN-restrict-SN-supt)*
qed

lemma *nrrstep-imp-pos-term*:
 $(Fun \ f \ ss, t) \in nrrstep \ R \implies$
 $\exists i \ s. t = Fun \ f \ (ss[i:=s]) \wedge (ss!i, s) \in rstep \ R \wedge i < length \ ss$
proof –
assume $(Fun \ f \ ss, t) \in nrrstep \ R$
then **obtain** $l \ r \ i \ ps \ \sigma$ **where** *rstep-rps: (Fun f ss, t) ∈ rstep-r-p-s R (l,r) (i#ps)*
 σ
unfolding *nrrstep-def* **by** *auto*
then **obtain** *C*
where $(l,r) \in R$
and $pos: (i\#ps) \in poss \ (Fun \ f \ ss)$

and $C: C = \text{ctx-of-pos-term } (i \# ps) \text{ (Fun } f \text{ ss)}$
and $C\langle l \cdot \sigma \rangle = \text{Fun } f \text{ ss}$
and $t: C\langle r \cdot \sigma \rangle = t$
unfolding $\text{rstep-r-p-s-def Let-def}$ **by** *auto*
then obtain D **where** $C = \text{More } f \text{ (take } i \text{ ss) } D \text{ (drop (Suc } i) \text{ ss)}$ **by** *auto*
with t **have** $t': t = \text{Fun } f \text{ (take } i \text{ ss @ (D}\langle r \cdot \sigma \rangle) \# \text{drop (Suc } i) \text{ ss)}$ **by** *auto*
from rstep-rps **have** $(ss!i, t[-i]) \in \text{rstep-r-p-s } R \text{ (l, r) ps } \sigma$
by $(\text{rule rstep-i-pos-imp-rstep-arg-i-pos})$
then have $\text{rstep:}(ss!i, t[-i]) \in \text{rstep } R$ **by** $(\text{auto simp: rstep-iff-rstep-r-p-s})$
then have $(C\langle ss!i \rangle, C\langle t[-i] \rangle) \in \text{rstep } R$ **..**
from pos **have** $\text{len: } i < \text{length ss}$ **by** *auto*
from len
have $(\text{take } i \text{ ss @ (D}\langle r \cdot \sigma \rangle) \# \text{drop (Suc } i) \text{ ss})!i = D\langle r \cdot \sigma \rangle$ **by** $(\text{auto simp: nth-append})$
with $C \text{ } t'$ **have** $t[-i] = D\langle r \cdot \sigma \rangle$ **by** *auto*
with t' **have** $t = \text{Fun } f \text{ (take } i \text{ ss @ (t[-i]) \# drop (Suc } i) \text{ ss)}$ **by** *auto*
with len **have** $t = \text{Fun } f \text{ (ss[i:=t[-i]])}$ **by** $(\text{auto simp: upd-conv-take-nth-drop})$
with rstep len **show** $\exists i \text{ s. } t = \text{Fun } f \text{ (ss[i:=s])} \wedge (ss!i, s) \in \text{rstep } R \wedge i < \text{length ss}$ **by** *auto*
qed

lemma $\text{rstep-cases}[\text{consumes } 1, \text{case-names root nonroot}]$:
 $\llbracket (s, t) \in \text{rstep } R; (s, t) \in \text{rrstep } R \implies P; (s, t) \in \text{nrrstep } R \implies P \rrbracket \implies P$
by $(\text{auto simp: rstep-iff-rrstep-or-nrrstep})$

lemma $\text{nrrstep-imp-rstep: } (s, t) \in \text{nrrstep } R \implies (s, t) \in \text{rstep } R$
by $(\text{auto simp: rstep-iff-rrstep-or-nrrstep})$

lemma $\text{nrrstep-imp-Fun: } (s, t) \in \text{nrrstep } R \implies \exists f \text{ ss. } s = \text{Fun } f \text{ ss}$
proof –
assume $(s, t) \in \text{nrrstep } R$
then obtain $i \text{ ps}$ **where** $i \# ps \in \text{poss } s$
unfolding $\text{nrrstep-def rstep-r-p-s-def Let-def}$ **by** *auto*
then show $\exists f \text{ ss. } s = \text{Fun } f \text{ ss}$ **by** $(\text{cases } s)$ *auto*
qed

lemma $\text{nrrstep-imp-subt-rstep}$:
assumes $(s, t) \in \text{nrrstep } R$
shows $\exists i. i < \text{num-args } s \wedge \text{num-args } s = \text{num-args } t \wedge (s[-[i], t[-[i]]) \in \text{rstep } R$
 $\wedge (\forall j. i \neq j \longrightarrow s[-[j] = t[-[j]])$
proof –
from $\langle (s, t) \in \text{nrrstep } R \rangle$ **obtain** $f \text{ ss}$ **where** $s = \text{Fun } f \text{ ss}$ **using** nrrstep-imp-Fun
by *blast*
with $\langle (s, t) \in \text{nrrstep } R \rangle$ **have** $(\text{Fun } f \text{ ss}, t) \in \text{nrrstep } R$ **by** *simp*
then obtain $i \text{ u}$ **where** $t = \text{Fun } f \text{ (ss[i := u])}$ **and** $(ss!i, u) \in \text{rstep } R$ **and** $i < \text{length ss}$
using $\text{nrrstep-imp-pos-term}$ **by** *best*
from $\langle s = \text{Fun } f \text{ ss} \rangle$ **and** $\langle t = \text{Fun } f \text{ (ss[i := u])} \rangle$ **have** $\text{num-args } s = \text{num-args } t$ **by** *auto*

from $\langle i < \text{length } ss \rangle$ **and** $\langle s = \text{Fun } f \text{ } ss \rangle$ **have** $i < \text{num-args } s$ **by** *auto*
from $\langle s = \text{Fun } f \text{ } ss \rangle$ **have** $s[-i] = (ss!i)$ **by** *auto*
from $\langle t = \text{Fun } f \text{ } (ss[i := u]) \rangle$ **and** $\langle i < \text{length } ss \rangle$ **have** $t[-i] = u$ **by** *auto*
from $\langle s = \text{Fun } f \text{ } ss \rangle$ **and** $\langle t = \text{Fun } f \text{ } (ss[i := u]) \rangle$
have $\forall j. j \neq i \longrightarrow s[-j] = t[-j]$ **by** *auto*
with $\langle (ss!i, u) \in \text{rstep } R \rangle$
and $\langle i < \text{num-args } s \rangle$
and $\langle \text{num-args } s = \text{num-args } t \rangle$
and $\langle s[-i] = (ss!i) \rangle[\text{symmetric}]$ **and** $\langle t[-i] = u \rangle[\text{symmetric}]$
show *?thesis* **by** (*auto*)
qed

lemma *finite-range*:

fixes $f :: \text{nat} \Rightarrow 'a$
assumes *finite (range f)*
shows $\exists x. \exists_{\infty} i. f i = x$
proof (*rule ccontr*)
assume $\neg(\exists x. \exists_{\infty} i. f i = x)$
then have $\forall x. \exists j. \forall i > j. f i \neq x$ **unfolding** *INFM-nat* **by** *blast*
from *choice[OF this]* **obtain** j **where** $\text{neq} : \forall x. \forall i > j. f i \neq x$..
from *finite-range-imageI[OF assms]* **have** *finite (range (j o f))* **by** (*simp add: comp-def*)
from *finite-nat-bounded[OF this]* **obtain** m **where** $\text{range } (j \circ f) \subseteq \{..<m\}$..
then have $j (f m) < m$ **by** (*auto simp: comp-def*)
with *neq* **show** *False* **by** *blast*
qed

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

proof –

from *assms* **obtain** $l \ r \ C \ \sigma$ **where** $(l, r) \in R$ **and** $C \neq \square$
and $s = C \langle l \cdot \sigma \rangle$ **and** $t = C \langle r \cdot \sigma \rangle$ **unfolding** *nrrstep-def'* **by** *best*
from $\langle C \neq \square \rangle$ s **have** $s \triangleright l \cdot \sigma$ **by** *auto*
moreover from $\langle C \neq \square \rangle$ t **have** $t \triangleright r \cdot \sigma$ **by** *auto*
moreover from $\langle (l, r) \in R \rangle$ **have** $(l \cdot \sigma, r \cdot \sigma) \in \text{rstep } R$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *nrrstep-args*:

assumes $(s, t) \in \text{nrrstep } R$
shows $\exists f \ ss \ ts. s = \text{Fun } f \ ss \wedge t = \text{Fun } f \ ts \wedge \text{length } ss = \text{length } ts$
 $\wedge (\exists j < \text{length } ss. (ss!j, ts!j) \in \text{rstep } R \wedge (\forall i < \text{length } ss. i \neq j \longrightarrow ss!i = ts!i))$
proof –
from *assms* **obtain** $l \ r \ C \ \sigma$ **where** $(l, r) \in R$ **and** $C \neq \square$
and $s = C \langle l \cdot \sigma \rangle$ **and** $t = C \langle r \cdot \sigma \rangle$ **unfolding** *nrrstep-def'* **by** *best*
from $\langle C \neq \square \rangle$ **obtain** $f \ ss1 \ D \ ss2$ **where** $C : C = \text{More } f \ ss1 \ D \ ss2$ **by** (*induct C*) *auto*
have $s = \text{Fun } f \ (ss1 @ D \langle l \cdot \sigma \rangle \# ss2)$ (*is - = Fun f ?ss*) **by** (*simp add: s C*)
moreover have $t = \text{Fun } f \ (ss1 @ D \langle r \cdot \sigma \rangle \# ss2)$ (*is - = Fun f ?ts*) **by** (*simp*)

add: $t \ C$
 moreover have $\text{length } ?ss = \text{length } ?ts$ by simp
 moreover have $\exists j < \text{length } ?ss.$
 $(?ss!j, ?ts!j) \in \text{rstep } R \wedge (\forall i < \text{length } ?ss. i \neq j \longrightarrow ?ss!i = ?ts!i)$
 proof -
 let $?j = \text{length } ss1$
 have $?j < \text{length } ?ss$ by simp
 moreover have $(?ss! ?j, ?ts! ?j) \in \text{rstep } R$
 proof -
 from $\langle l, r \rangle \in R$ have $(D\langle l \cdot \sigma \rangle, D\langle r \cdot \sigma \rangle) \in \text{rstep } R$ by auto
 then show ?thesis by auto
 qed
 moreover have $\forall i < \text{length } ?ss. i \neq ?j \longrightarrow ?ss!i = ?ts!i$ (is $\forall i < \text{length } ?ss. -$
 $\longrightarrow ?P \ i$)
 proof (intro allI impI)
 fix i assume $i < \text{length } ?ss$ and $i \neq ?j$
 then have $i < \text{length } ss1 \vee \text{length } ss1 < i$ by auto
 then show ?P i
 proof
 assume $i < \text{length } ss1$ then show ?P i by (auto simp: nth-append)
 next
 assume $i > \text{length } ss1$ then show ?P i
 using $\langle i < \text{length } ?ss \rangle$ by (auto simp: nth-Cons' nth-append)
 qed
 qed
 ultimately show ?thesis by best
 qed
 ultimately show ?thesis by auto
 qed

 lemma nrrstep-iff-arg-rstep:
 $(s, t) \in \text{nrrstep } R \longleftrightarrow$
 $(\exists f \ ss \ i \ t'. s = \text{Fun } f \ ss \ t = \text{Fun } f \ ts \wedge i < \text{length } ss \wedge t = \text{Fun } f \ (ss[i:=t']) \wedge (ss!i, t') \in$
 $\text{rstep } R)$
 (is $?L \longleftrightarrow ?R$)
 proof
 assume $L: ?L$
 from nrrstep-args[OF this]
 obtain $f \ ss \ ts$ where $s = \text{Fun } f \ ss \ t = \text{Fun } f \ ts$ by auto
 with nrrstep-imp-pos-term[OF L[unfolded this]]
 show ?R by auto
 next assume $R: ?R$
 then obtain $f \ i \ ss \ t'$
 where $s: s = \text{Fun } f \ ss$ and $t: t = \text{Fun } f \ (ss[i:=t'])$
 and $i: i < \text{length } ss$ and $st': (ss ! i, t') \in \text{rstep } R$ by auto
 from st' obtain $C \ l \ r \ \sigma$ where $lr: \langle l, r \rangle \in R$ and $s': ss!i = C\langle l \cdot \sigma \rangle$ and $t': t' = C\langle r \cdot \sigma \rangle$ by auto
 let $?D = \text{More } f \ (\text{take } i \ ss) \ C \ (\text{drop } (\text{Suc } i) \ ss)$
 have $s = ?D\langle l \cdot \sigma \rangle \ t = ?D\langle r \cdot \sigma \rangle$ unfolding $s \ t$

using *id-take-nth-drop*[*OF i*] *upd-conv-take-nth-drop*[*OF i*] *s' t'* by *auto*
 with *lr* show ?*L* apply(*rule nrrstepI*) using *t'* by *auto*
 qed

lemma *subterms-NF-imp-SN-on-nrrstep*:

assumes $\forall s \triangleleft t. s \in NF \text{ (rstep } R)$ shows *SN-on* (*nrrstep* *R*) {*t*}

proof

fix *S* assume *S* 0 $\in \{t\}$ and $\forall i. (S \ i, S \ (Suc \ i)) \in \text{nrrstep } R$

then have $(t, S \ (Suc \ 0)) \in \text{nrrstep } R$ by *auto*

then obtain *l r C* σ where $(l, r) \in R$ and $C \neq \square$ and $t: t = C \langle l \cdot \sigma \rangle$ unfolding *nrrstep-def'* by *auto*

then obtain *f ss1 D ss2* where $C: C = \text{More } f \text{ ss1 } D \text{ ss2}$ by (*induct* *C*) *auto*

have $t \triangleright D \langle l \cdot \sigma \rangle$ unfolding *t C* by *auto*

moreover have $D \langle l \cdot \sigma \rangle \notin NF \text{ (rstep } R)$

proof –

from $\langle (l, r) \in R \rangle$ have $(D \langle l \cdot \sigma \rangle, D \langle r \cdot \sigma \rangle) \in \text{rstep } R$ by *auto*

then show ?*thesis* by *auto*

qed

ultimately show *False* using *assms* by *simp*

qed

lemma *args-NF-imp-SN-on-nrrstep*:

assumes $\forall t \in \text{set } ts. t \in NF \text{ (rstep } R)$ shows *SN-on* (*nrrstep* *R*) {*Fun f ts*}

proof

fix *S* assume *S* 0 $\in \{\text{Fun } f \text{ ts}\}$ and $\forall i. (S \ i, S \ (Suc \ i)) \in \text{nrrstep } R$

then have $(\text{Fun } f \text{ ts}, S \ (Suc \ 0)) \in \text{nrrstep } R$

unfolding *singletonD*[*OF* $\langle S \ 0 \in \{\text{Fun } f \text{ ts}\} \rangle$, *symmetric*] by *simp*

then obtain *l r C* σ where $(l, r) \in R$ and $C \neq \square$ and $\text{Fun } f \text{ ts} = C \langle l \cdot \sigma \rangle$

unfolding *nrrstep-def'* by *auto*

then obtain *ss1 D ss2* where $C: C = \text{More } f \text{ ss1 } D \text{ ss2}$ by (*induct* *C*) *auto*

with $\langle \text{Fun } f \text{ ts} = C \langle l \cdot \sigma \rangle \rangle$ have $D \langle l \cdot \sigma \rangle \in \text{set } ts$ by *auto*

moreover have $D \langle l \cdot \sigma \rangle \notin NF \text{ (rstep } R)$

proof –

from $\langle (l, r) \in R \rangle$ have $(D \langle l \cdot \sigma \rangle, D \langle r \cdot \sigma \rangle) \in \text{rstep } R$ by *auto*

then show ?*thesis* by *auto*

qed

ultimately show *False* using *assms* by *simp*

qed

lemma *rrstep-imp-rule-subst*:

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

shows $\exists l \ r \ \sigma. (l, r) \in R \wedge (l \cdot \sigma) = s \wedge (r \cdot \sigma) = t$

proof –

have *ctxt-of-pos-term* $\square \ s = \text{Hole}$ by *auto*

obtain *l r* σ where $(s, t) \in \text{rstep-r-p-s } R \ (l, r) \ \square \ \sigma$ using *assms* unfolding

rrstep-def by *best*

then have let $C = \text{ctxt-of-pos-term } \square \ s$ in $\square \in \text{poss } s \wedge (l, r) \in R \wedge C \langle l \cdot \sigma \rangle = s$
 $\wedge C \langle r \cdot \sigma \rangle = t$ unfolding *rstep-r-p-s-def* by *simp*

with $\langle \text{ctxt-of-pos-term } [] \ s = \text{Hole} \rangle$ **have** $(l,r) \in R$ **and** $l \cdot \sigma = s$ **and** $r \cdot \sigma = t$
unfolding *Let-def* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *nrrstep-preserves-root*:

assumes $(\text{Fun } f \ ss, t) \in \text{nrrstep } R$ **(is** $(?s, t) \in \text{nrrstep } R$) **shows** $\exists ts. t = (\text{Fun } f \ ts)$
using *assms* **unfolding** *nrrstep-def rstep-r-p-s-def Let-def* **by** *auto*

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

proof –

from *assms* **obtain** $f \ ss$ **where** $s = \text{Fun } f \ ss$ **using** *nrrstep-imp-Fun* **by** *best*
with *assms* **obtain** ts **where** $t = \text{Fun } f \ ts$ **using** *nrrstep-preserves-root* **by** *best*
from $\langle s = \text{Fun } f \ ss \rangle$ **and** $\langle t = \text{Fun } f \ ts \rangle$ **show** *?thesis* **by** *best*
qed

lemma *nrrstep-reflects-root*:

assumes $(s, \text{Fun } g \ ts) \in \text{nrrstep } R$ **(is** $(s, ?t) \in \text{nrrstep } R$)
shows $\exists ss. s = (\text{Fun } g \ ss)$

proof –

from *assms* **obtain** $f \ ss \ ts'$ **where** $s = \text{Fun } f \ ss$ **and** $\text{Fun } g \ ts = \text{Fun } f \ ts'$ **using** *nrrstep-equiv-root* **by** *best*
then have $f = g$ **by** *simp*
with $\langle s = \text{Fun } f \ ss \rangle$ **have** $s = \text{Fun } g \ ss$ **by** *simp*
then show *?thesis* **by** *auto*
qed

lemma *nrrsteps-preserve-root*:

assumes $(\text{Fun } f \ ss, t) \in (\text{nrrstep } R)^*$
shows $\exists ts. t = (\text{Fun } f \ ts)$
using *assms* **by** *induct (auto simp: nrrstep-preserves-root)*

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

assumes $(\text{Fun } f \ ss, \text{Fun } f \ ts) \in \text{nrrstep } R$ **(is** $(?s, ?t) \in \text{nrrstep } R$) **and** $i < \text{length } ss$

shows $(ss!i, ts!i) \in (\text{rstep } R)^*$

proof –

from *assms* **have** $[i] \in \text{poss } ?s$ **using** *empty-pos-in-poss* **by** *simp*
from $\langle (?s, ?t) \in \text{nrrstep } R \rangle$
obtain $l \ r \ j \ ps \ \sigma$
where A : **let** $C = \text{ctxt-of-pos-term } (j \# ps) \ ?s$ **in** $(j \# ps) \in \text{poss } ?s \wedge (l, r) \in R \wedge C \langle l \cdot \sigma \rangle = ?s \wedge C \langle r \cdot \sigma \rangle = ?t$ **unfolding** *nrrstep-def rstep-r-p-s-def* **by** *force*
let $?C = \text{ctxt-of-pos-term } (j \# ps) \ ?s$
from A **have** $(j \# ps) \in \text{poss } ?s$ **and** $(l, r) \in R$ **and** $?C \langle l \cdot \sigma \rangle = ?s$ **and** $?C \langle r \cdot \sigma \rangle = ?t$ **using** *Let-def* **by** *auto*
have C : $?C = \text{More } f \ (\text{take } j \ ss) \ (\text{ctxt-of-pos-term } ps \ (ss!j)) \ (\text{drop } (\text{Suc } j) \ ss)$ **(is** $?C = \text{More } f \ ?ss1 \ ?D \ ?ss2$) **by** *auto*

```

from  $\langle ?C\langle l.\sigma \rangle = ?s \rangle$  have  $?D\langle l.\sigma \rangle = (ss!j)$  by (auto simp: take-drop-imp-nth)
from  $\langle (l,r) \in R \rangle$  have  $(l.\sigma, r.\sigma) \in (\text{subst.closure } R)$  by auto
then have  $(?D\langle l.\sigma \rangle, ?D\langle r.\sigma \rangle) \in (\text{ctxt.closure}(\text{subst.closure } R))$ 
  and  $(?C\langle l.\sigma \rangle, ?C\langle r.\sigma \rangle) \in (\text{ctxt.closure}(\text{subst.closure } R))$  by (auto simp only:
ctxt.closure.intros)
then have  $D\text{-rstep}: (?D\langle l.\sigma \rangle, ?D\langle r.\sigma \rangle) \in \text{rstep } R$  and  $(?C\langle l.\sigma \rangle, ?C\langle r.\sigma \rangle) \in \text{rstep}$ 
R
  unfolding rstep-eq-closure by auto
from  $\langle ?C\langle r.\sigma \rangle = ?t \rangle$  and C have  $?t = \text{Fun } f \text{ (take } j \text{ ss @ } ?D\langle r.\sigma \rangle \# \text{drop (Suc } j) \text{ ss)}$  by auto
then have  $ts: ts = (\text{take } j \text{ ss @ } ?D\langle r.\sigma \rangle \# \text{drop (Suc } j) \text{ ss})$  by auto
have  $j = i \vee j \neq i$  by simp
from  $\langle j \# ps \in \text{poss } ?s \rangle$  have  $j < \text{length } ss$  by simp
then have  $(\text{take } j \text{ ss @ } ?D\langle r.\sigma \rangle \# \text{drop (Suc } j) \text{ ss}) ! j = ?D\langle r.\sigma \rangle$  by (auto simp:
nth-append-take)
with ts have  $ts!j = ?D\langle r.\sigma \rangle$  by auto
have  $j = i \vee j \neq i$  by simp
then show  $(ss!i, ts!i) \in (\text{rstep } R)^*$ 
proof
  assume  $j = i$ 
  with  $\langle ts!j = ?D\langle r.\sigma \rangle \rangle$  and  $\langle ?D\langle l.\sigma \rangle = ss!j \rangle$  and D-rstep show ?thesis by auto
next
  assume  $j \neq i$ 
  with  $\langle i < \text{length } ss \rangle$  and  $\langle j < \text{length } ss \rangle$  have  $(\text{take } j \text{ ss @ } ?D\langle r.\sigma \rangle \# \text{drop (Suc } j) \text{ ss}) ! i = ss!i$  by (auto simp: nth-append-take-drop-is-nth-conv)
  with ts have  $ts!i = ss!i$  by auto
  then show ?thesis by auto
qed
qed

lemma nrrstep-imp-arg-rsteps:
  assumes  $(s,t) \in \text{nrrstep } R$  and  $i < \text{num-args } s$  shows  $(\text{args } s!i, \text{args } t!i) \in (\text{rstep } R)^*$ 
proof (cases s)
  fix x assume  $s = \text{Var } x$  then show ?thesis using assms by auto
next
  fix f ss assume  $s = \text{Fun } f \text{ ss}$ 
  then have  $(\text{Fun } f \text{ ss}, t) \in \text{nrrstep } R$  using assms by simp
  then obtain ts where  $t = \text{Fun } f \text{ ts}$  using nrrstep-preserves-root by best
  with  $\langle (s,t) \in \text{nrrstep } R \rangle$  and  $\langle s = \text{Fun } f \text{ ss} \rangle$  have  $(\text{Fun } f \text{ ss}, \text{Fun } f \text{ ts}) \in \text{nrrstep}$ 
R by simp
  from  $\langle s = \text{Fun } f \text{ ss} \rangle$  and  $\langle i < \text{num-args } s \rangle$  have  $i < \text{length } ss$  by simp
  with  $\langle (\text{Fun } f \text{ ss}, \text{Fun } f \text{ ts}) \in \text{nrrstep } R \rangle$ 
    have  $(ss!i, ts!i) \in (\text{rstep } R)^*$  by (rule nrrstep-Fun-imp-arg-rsteps)
  with  $\langle s = \text{Fun } f \text{ ss} \rangle$  and  $\langle t = \text{Fun } f \text{ ts} \rangle$  show ?thesis by simp
qed

```

```

lemma nrrsteps-imp-rsteps:  $(s,t) \in (\text{nrrstep } R)^* \implies (s,t) \in (\text{rstep } R)^*$ 
proof (induct rule: rtrancl.induct)

```

case (rtranc1-refl a) then show ?case by simp
 next
 case (rtranc1-into-rtranc1 a b c)
 then have IH: $(a,b) \in (rstep\ R)^*$ and nrrstep: $(b,c) \in nrrstep\ R$ by auto
 from nrrstep have $(b,c) \in rstep\ R$ using nrrstep-imp-rstep by auto
 with IH show ?case by auto
 qed

lemma nrrstep-Fun-preserves-num-args:
 assumes $(Fun\ f\ ss, Fun\ f\ ts) \in nrrstep\ R$ (is $(?s, ?t) \in nrrstep\ R$)
 shows $length\ ss = length\ ts$
proof –
 from assms obtain $l\ r\ i\ ps\ \sigma$
 where let $C = ctxt\ of\ pos\ term\ (i\#ps)\ ?s$ in $(i\#ps) \in poss\ ?s \wedge (l,r) \in R \wedge$
 $C\langle l.\sigma \rangle = ?s \wedge C\langle r.\sigma \rangle = ?t$ (is let $C = ?C$ in -)
 unfolding nrrstep-def rstep-r-p-s-def by force
 then have $(l,r) \in R$ and $Cl: ?C\langle l.\sigma \rangle = ?s$ and $Cr: ?C\langle r.\sigma \rangle = ?t$ unfolding
 Let-def by auto
 have $C: ?C = More\ f\ (take\ i\ ss)\ (ctxt\ of\ pos\ term\ ps\ (ss!i))\ (drop\ (Suc\ i)\ ss)$ (is
 $?C = More\ f\ ?ss1\ ?D\ ?ss2$) by simp
 from C and Cl have $s: ?s = Fun\ f\ (take\ i\ ss\ @\ ?D\langle l.\sigma \rangle\ \# drop\ (Suc\ i)\ ss)$ (is
 $?s = Fun\ f\ ?ss$) by simp
 from C and Cr have $t: ?t = Fun\ f\ (take\ i\ ss\ @\ ?D\langle r.\sigma \rangle\ \# drop\ (Suc\ i)\ ss)$ (is
 $?t = Fun\ f\ ?ts$) by simp
 from s and t have $ss: ss = ?ss$ and $ts: ts = ?ts$ by auto
 have $length\ ?ss = length\ ?ts$ by auto
 with ss and ts show ?thesis by simp
 qed

lemma nrrstep-equiv-num-args:
 assumes $(s,t) \in nrrstep\ R$ shows $num\ args\ s = num\ args\ t$
proof –
 from assms obtain $f\ ss\ ts$ where $s:s = Fun\ f\ ss$ and $t:t = Fun\ f\ ts$ using
 nrrstep-equiv-root by best
 with assms have $(Fun\ f\ ss, Fun\ f\ ts) \in nrrstep\ R$ by simp
 then have $length\ ss = length\ ts$ by (rule nrrstep-Fun-preserves-num-args)
 with s and t show ?thesis by auto
 qed

lemma nrrsteps-equiv-num-args:
 assumes $(s,t) \in (nrrstep\ R)^*$ shows $num\ args\ s = num\ args\ t$
 using assms by (induct, auto simp: nrrstep-equiv-num-args)

lemma nrrstep-preserves-num-args:
 assumes $(s,t) \in nrrstep\ R$ and $i < num\ args\ s$ shows $i < num\ args\ t$
proof (cases s)
 fix x assume $s = Var\ x$ then show ?thesis using assms by auto
 next
 fix f ss assume $s = Fun\ f\ ss$

with *assms* obtain *ts* where $t = \text{Fun } f \text{ } ts$ using *nrrstep-preserves-root* by best
 from $\langle (s,t) \in \text{nrrstep } R \rangle$ have $\text{length } ss = \text{length } ts$ unfolding $\langle s = \text{Fun } f \text{ } ss \rangle$
 and $\langle t = \text{Fun } f \text{ } ts \rangle$ by (rule *nrrstep-Fun-preserves-num-args*)
 with *assms* and $\langle s = \text{Fun } f \text{ } ss \rangle$ and $\langle t = \text{Fun } f \text{ } ts \rangle$ show ?thesis by auto
 qed

lemma *nrrstep-reflects-num-args*:
 assumes $(s,t) \in \text{nrrstep } R$ and $i < \text{num-args } t$ shows $i < \text{num-args } s$
 proof (cases *t*)
 fix *x* assume $t = \text{Var } x$ then show ?thesis using *assms* by auto
 next
 fix *g ts* assume $t = \text{Fun } g \text{ } ts$
 with *assms* obtain *ss* where $s = \text{Fun } g \text{ } ss$ using *nrrstep-reflects-root* by best
 from $\langle (s,t) \in \text{nrrstep } R \rangle$ have $\text{length } ss = \text{length } ts$ unfolding $\langle s = \text{Fun } g \text{ } ss \rangle$
 and $\langle t = \text{Fun } g \text{ } ts \rangle$ by (rule *nrrstep-Fun-preserves-num-args*)
 with *assms* and $\langle s = \text{Fun } g \text{ } ss \rangle$ and $\langle t = \text{Fun } g \text{ } ts \rangle$ show ?thesis by auto
 qed

lemma *nrrsteps-imp-arg-rsteps*:
 assumes $(s,t) \in (\text{nrrstep } R)^*$ and $i < \text{num-args } s$
 shows $(\text{args } s!i, \text{args } t!i) \in (\text{rstep } R)^*$
 using *assms*
 proof (induct rule: *rtrancl.induct*)
 case (*rtrancl-refl a*) then show ?case by auto
 next
 case (*rtrancl-into-rtrancl a b c*)
 then have *IH*: $(\text{args } a!i, \text{args } b!i) \in (\text{rstep } R)^*$ by auto
 from $\langle (a,b) \in (\text{nrrstep } R)^* \rangle$ and $i < \text{num-args } a$ have $i < \text{num-args } b$ by
 induct (auto simp: *nrrstep-preserves-num-args*)
 with $\langle (b,c) \in \text{nrrstep } R \rangle$
 have $(\text{args } b!i, \text{args } c!i) \in (\text{rstep } R)^*$ by (rule *nrrstep-imp-arg-rsteps*)
 with *IH* show ?case by simp
 qed

lemma *nrrsteps-imp-eq-root-arg-rsteps*:
 assumes $\text{steps: } (s,t) \in (\text{nrrstep } R)^*$
 shows $\text{root } s = \text{root } t \wedge (\forall i < \text{num-args } s. (s \mid - [i], t \mid - [i]) \in (\text{rstep } R)^*)$
 proof (cases *s*)
 case (*Var x*)
 have $s = t$ using *steps* unfolding *Var*
 proof (induct)
 case (*step y z*)
 from *nrrstep-imp-Fun*[*OF step*(2)] *step*(3) have *False* by auto
 then show ?case ..
 qed *simp*
 then show ?thesis by auto
 next
 case (*Fun f ss*)
 from *nrrsteps-equiv-num-args*[*OF steps*]

$nrrsteps\text{-}imp\text{-}arg\text{-}rsteps[OF\ steps]$
 $nrrsteps\text{-}preserve\text{-}root[OF\ steps[unfolded\ Fun]]$
show *?thesis* **unfolding** *Fun* **by** *auto*
qed

definition

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

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

$wf\text{-}trs\ R \Longrightarrow (l,r) \in R \Longrightarrow \exists\ f\ ts.\ l = Fun\ f\ ts$
unfolding *wf-trs-def* **by** *blast*

lemma *rstep-imp-Fun*:

assumes *wf-trs R*
shows $(s, t) \in rstep\ R \Longrightarrow \exists\ f\ ss.\ s = Fun\ f\ ss$
proof –
assume $(s, t) \in rstep\ R$
then obtain $C\ l\ r\ \sigma$ **where** $lr: (l,r) \in R$ **and** $s: s = C\ \langle l \cdot \sigma \rangle$ **by** *auto*
with *wf-trs-imp-lhs-Fun[OF assms lr]* **show** *?thesis* **by** (*cases C, auto*)
qed

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

assumes *SN-on (rstep R) {t}* **shows** $\forall\ s \sqsubseteq t.\ SN\text{-on}\ (rstep\ R)\ \{s\}$
proof (*rule ccontr*)
assume $\neg(\forall\ s \sqsubseteq t.\ SN\text{-on}\ (rstep\ R)\ \{s\})$
then obtain s **where** $t \sqsupseteq s$ **and** $\neg SN\text{-on}\ (rstep\ R)\ \{s\}$ **by** *auto*
then obtain S **where** $S\ 0 = s$ **and** *chain: chain (rstep R) S* **by** *auto*
from $\langle t \sqsupseteq s \rangle$ **obtain** C **where** $t: t = C\langle s \rangle$ **by** *auto*
let $?S = \lambda i.\ C\langle S\ i \rangle$
from $\langle S\ 0 = s \rangle$ **have** $?S\ 0 = t$ **by** (*simp add: t*)
moreover from *chain* **have** *chain (rstep R) ?S* **by** *blast*
ultimately have $\neg SN\text{-on}\ (rstep\ R)\ \{t\}$ **by** *best*
with *assms* **show** *False* **by** *simp*
qed

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

assumes $\neg SN\text{-on}\ (rstep\ R)\ \{t\}$ **and** $s \sqsupseteq t$
shows $\neg SN\text{-on}\ (rstep\ R)\ \{s\}$
using *assms* *SN-on-imp-SN-on-subt* **by** *blast*

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

assumes *SN-on (rstep R) {t · σ}* **and** $x \in vars\text{-}term\ t$
shows *SN-on (rstep R) {Var x · σ}*
proof –
from *assms* **have** $t \sqsupseteq Var\ x$ **by** *auto*
then have $t \cdot \sigma \sqsupseteq (Var\ x) \cdot \sigma$ **by** (*rule supseq-subst*)

with *SN-on-imp-SN-on-subt* and *assms* show *?thesis* by *best*
qed

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

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

proof –

from *assms* have $x \in \bigcup (\text{set } (\text{map vars-term ss}))$ by *simp*
then have $x \in (\bigcup i < \text{length ss}. \text{vars-term}(ss!i))$ unfolding *set-conv-nth* by *auto*
then have $\exists i < \text{length ss}. x \in \text{vars-term}(ss!i)$ using *UN-iff* by *best*
then obtain *i* where $i < \text{length ss}$ and $x \in \text{vars-term}(ss!i)$ by *auto*
then have $i < \text{num-args } ?s$ by *simp*
with $\langle x \in \text{vars-term}(ss!i) \rangle$ show *?thesis* by *auto*

qed

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

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

proof (induct *s* arbitrary: *t* rule: *term.induct*)

case (*Var x*) then show *?case* by *auto*

next

case (*Fun f ss*)

from $\langle \text{Fun } f \text{ ss} \cdot \sigma \triangleright t \rangle$ have $(\text{Fun } f \text{ ss} \cdot \sigma = t) \vee (\text{Fun } f \text{ ss} \cdot \sigma \triangleright t)$ by *auto*

then show *?case*

proof

assume $\text{Fun } f \text{ ss} \cdot \sigma = t$ with *Fun* show *?thesis* by *auto*

next

assume $\text{Fun } f \text{ ss} \cdot \sigma \triangleright t$

then have $\text{Fun } f (\text{map } (\lambda t. t \cdot \sigma) \text{ ss}) \triangleright t$ by *simp*

then have $\exists s \in \text{set } (\text{map } (\lambda t. t \cdot \sigma) \text{ ss}). s \triangleright t$ (is $\exists s \in \text{set } ?ss'. s \triangleright t$) by (rule *supt-Fun-imp-arg-supteq*)

then obtain *s'* where $s' \in \text{set } ?ss'$ and $s' \triangleright t$ by *best*

then have $\exists i < \text{length } ?ss'. ?ss'!i = s'$ using *in-set-conv-nth*[where $?x = s'$]
by *best*

then obtain *i* where $i < \text{length } ?ss'$ and $?ss'!i = s'$ by *best*

then have $?ss'!i = (ss!i) \cdot \sigma$ by *auto*

from $\langle ?ss'!i = s' \rangle$ have $s' = (ss!i) \cdot \sigma$ unfolding $\langle ?ss'!i = (ss!i) \cdot \sigma \rangle$ by *simp*

from $\langle s' \triangleright t \rangle$ have $(ss!i) \cdot \sigma \triangleright t$ unfolding $\langle s' = (ss!i) \cdot \sigma \rangle$ by *simp*

with $\langle i < \text{length } ?ss' \rangle$ have $(ss!i) \in \text{set } ss$ by *auto*

with $\langle (ss!i) \cdot \sigma \triangleright t \rangle$ have $\exists s \in \text{set } ss. s \cdot \sigma \triangleright t$ by *best*

then obtain *s* where $s \in \text{set } ss$ and $s \cdot \sigma \triangleright t$ by *best*

with *Fun* have $\forall x \in \text{vars-term } s. \neg((\text{Var } x) \cdot \sigma \triangleright t)$ by *force*

from *Fun*

have *IH*: $s \in \text{set } ss \longrightarrow (\forall v. s \cdot \sigma \triangleright v \longrightarrow (\forall x \in \text{vars-term } s. \neg \text{Var } x \cdot \sigma \triangleright v)) \longrightarrow (\exists u. s \triangleright u \wedge v = u \cdot \sigma)$

by *auto*

with $\langle s \in \text{set } ss \rangle$ have $!!v. s \cdot \sigma \triangleright v \implies (\forall x \in \text{vars-term } s. \neg \text{Var } x \cdot \sigma \triangleright v) \longrightarrow (\exists u. s \triangleright u \wedge v = u \cdot \sigma)$

by *simp*

with $\langle s \cdot \sigma \triangleright t \rangle$ have $(\forall x \in \text{vars-term } s. \neg \text{Var } x \cdot \sigma \triangleright t) \longrightarrow (\exists u. s \triangleright u \wedge t =$

$u \cdot \sigma$) **by** *simp*
with $\langle \forall x \in \text{vars-term } s. \neg \text{Var } x \cdot \sigma \supseteq t \rangle$ **obtain** u **where** $s \supseteq u$ **and** $t = u \cdot \sigma$
by *best*
with $\langle s \in \text{set } ss \rangle$ **have** $\text{Fun } f \ ss \supseteq u$ **by** *auto*
with $\langle t = u \cdot \sigma \rangle$ **show** *?thesis* **by** *best*
qed
qed

lemma *SN-imp-SN-subt*:
 $\text{SN-on } (\text{rstep } R) \{s\} \implies s \supseteq t \implies \text{SN-on } (\text{rstep } R) \{t\}$
by (rule *ctxt-closed-SN-on-subt*[*OF* *ctxt-closed-rstep*])

lemma *subterm-preserves-SN-gen*:
assumes *ctxt*: *ctxt.closed* R
and SN : $\text{SN-on } R \{t\}$ **and** *supt*: $t \triangleright s$
shows $\text{SN-on } R \{s\}$
proof –
from *supt* **have** $t \supseteq s$ **by** *auto*
then show *?thesis* **using** *ctxt-closed-SN-on-subt*[*OF* *ctxt* SN , *of* s] **by** *simp*
qed

context *E-compatible*
begin

lemma *SN-on-step-E-imp-SN-on*: **assumes** $\text{SN-on } R \{s\}$
and $(s, t) \in E$
shows $\text{SN-on } R \{t\}$
using *assms* $E(1)$ **unfolding** *SN-on-all-reducts-SN-on-conv*[*of* - t] *SN-on-all-reducts-SN-on-conv*[*of* - s]
by *blast*

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 : $\text{SN-on } R \{s\}$
shows $\text{SN-on } R \{t\}$
proof (*cases* $(s, t) \in R$)
case *True*
from *step-preserves-SN-on*[*OF* *this* SN] **show** *?thesis* .
next
case *False*
with st **obtain** $u \ v$ **where** su : $(s, u) \in E$ **and** uv : $u \triangleright v$ **and** vt : $(v, t) \in E$ **by** *auto*
have u : $\text{SN-on } R \{u\}$ **by** (rule *SN-on-step-E-imp-SN-on*[*OF* SN su])
with $uv \ R$ **have** $\text{SN-on } R \{v\}$ **by** (*metis* *subterm-preserves-SN-gen*)
then show *?thesis* **by** (rule *SN-on-step-E-imp-SN-on*[*OF* - vt])
qed

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

$ctxt.closed\ R \implies SN-on\ R\ \{s\} \implies (s,t) \in R \cup E\ O\ \{\triangleright\}\ O\ E \implies (s,t) \in restrict-SN-supt-E$

unfolding $restrict-SN-supt-E-def\ restrict-SN-def$
using $SN-on-step-REs-imp-SN-on[of\ s\ t]\ E(2)$ **by** *auto*

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

assumes $R: ctxt.closed\ R$
and $SN: SN\ (E\ O\ \{\triangleright\}\ O\ E)$
shows $SN-on\ (R \cup E\ O\ \{\triangleright\}\ O\ E)\ \{t.\ SN-on\ R\ \{t\}\}$
proof –
 {
 fix f
 assume $f0: SN-on\ R\ \{f\ 0\}$ **and** $f: \bigwedge i. (f\ i, f\ (Suc\ i)) \in R \cup E\ O\ \{\triangleright\}\ O\ E$
 from $ctxt-closed-imp-SN-restrict-SN-E-supt[OF\ assms]$
 have $SN: SN\ restrict-SN-supt-E$.
 {
 fix i
 have $SN-on\ R\ \{f\ i\}$
 by $(induct\ i, rule\ f0, rule\ SN-on-step-REs-imp-SN-on[OF\ R\ f])$
 } **note** $fi = this$
 {
 fix i
 from $f[of\ i]\ fi[of\ i]$
 have $(f\ i, f\ (Suc\ i)) \in restrict-SN-supt-E$ **by** $(metis\ restrict-SN-supt-E-I[OF\ R])$
 }
 with SN **have** $False$ **by** *auto*
 }
then show $?thesis$ **unfolding** $SN-on-def$ **by** *blast*
qed
end

lemma $subterm-preserves-SN$:

$SN-on\ (rstep\ R)\ \{t\} \implies t \triangleright s \implies SN-on\ (rstep\ R)\ \{s\}$
by $(rule\ subterm-preserves-SN-gen[OF\ ctxt-closed-rstep])$

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

assumes $ctxt: ctxt.closed\ R$
and $SN-on\ R\ T$
shows $SN-on\ (supt \cup R)\ T$ **(is** $SN-on\ ?S\ T$ **)**
proof $(rule\ ccontr)$
assume $\neg SN-on\ ?S\ T$
then obtain s **where** $ini: s\ 0 : T$ **and** $chain: chain\ ?S\ s$
unfolding $SN-on-def$ **by** *auto*
have $SN: \forall i. SN-on\ R\ \{s\ i\}$
proof
fix i **show** $SN-on\ R\ \{s\ i\}$

```

proof (induct i)
  case 0 show ?case using assms using  $\langle s \ 0 \in T \rangle$  and SN-on-subset2[of {s
0} T R] by simp
next
  case (Suc i)
  from chain have (s i, s (Suc i))  $\in$  ?S by simp
  then show ?case
  proof
    assume (s i, s (Suc i))  $\in$  supt
    from subterm-preserves-SN-gen[OF ctxt Suc this] show ?thesis .
  next
    assume (s i, s (Suc i))  $\in$  R
    from step-preserves-SN-on[OF this Suc] show ?thesis .
  qed
qed
qed
have  $\neg (\exists S. \forall i. (S \ i, S \ (Suc \ i)) \in \text{restrict-SN-supt } R)$ 
  using ctxt-closed-imp-SN-restrict-SN-supt[OF ctxt] unfolding SN-defs by auto
moreover have  $\forall i. (s \ i, s \ (Suc \ i)) \in \text{restrict-SN-supt } R$ 
proof
  fix i
  from SN have SN: SN-on R {s i} by simp
  from chain have (s i, s (Suc i))  $\in$  supt  $\cup$  R by simp
  then show (s i, s (Suc i))  $\in$  restrict-SN-supt R
    unfolding restrict-SN-supt-def restrict-SN-def using SN by auto
qed
ultimately show False by auto
qed

```

lemma *SN-on-rstep-imp-SN-on-supt-union-rstep*:
 $SN\text{-on } (rstep \ R) \ T \implies SN\text{-on } (supt \cup rstep \ R) \ T$
by (*rule SN-on-r-imp-SN-on-supt-union-r*[*OF ctxt-closed-rstep*])

lemma *SN-supt-r-trancl*:
assumes *ctxt*: *ctxt.closed R*
and *a*: *SN R*
shows *SN* ((*supt* \cup *R*)⁺)
proof –
have *SN* (*supt* \cup *R*)
using *SN-on-r-imp-SN-on-supt-union-r*[*OF ctxt, of UNIV*]
and *a* **by** *force*
then **show** *SN* ((*supt* \cup *R*)⁺) **by** (*rule SN-imp-SN-trancl*)
qed

lemma *SN-supt-rstep-trancl*:
 $SN \ (rstep \ R) \implies SN \ ((supt \cup rstep \ R)^+)$
by (*rule SN-supt-r-trancl*[*OF ctxt-closed-rstep*])

lemma *SN-imp-SN-arg-gen*:

assumes *ctxt*: *ctxt.closed R*
and *SN*: *SN-on R {Fun f ts}* **and** *arg*: $t \in \text{set } ts$ **shows** *SN-on R {t}*
proof –
from *arg* **have** *Fun f ts* $\supseteq t$ **by** *auto*
with *SN* **show** *?thesis* **by** (*rule ctxt-closed-SN-on-subt[OF ctxt]*)
qed

lemma *SN-imp-SN-arg*:
SN-on (rstep R) {Fun f ts} $\implies t \in \text{set } ts \implies \text{SN-on (rstep R) } \{t\}$
by (*rule SN-imp-SN-arg-gen[OF ctxt-closed-rstep]*)

lemma *SNinstance-imp-SN*:
assumes *SN-on (rstep R) {t · σ }*
shows *SN-on (rstep R) {t}*
proof
fix *f*
assume *prem*: $f \ 0 \in \{t\} \ \forall i. (f \ i, f \ (\text{Suc } i)) \in \text{rstep } R$
let $?g = \lambda i. (f \ i) \cdot \sigma$
from *prem* **have** $?g \ 0 = t \cdot \sigma \wedge (\forall i. (?g \ i, ?g \ (\text{Suc } i)) \in \text{rstep } R)$ **using**
subst-closed-rstep
by *auto*
then **have** $\neg \text{SN-on (rstep R) } \{t \cdot \sigma\}$ **by** *auto*
with *assms* **show** *False* **by** *blast*
qed

lemma *rstep-imp-C-s-r*:
assumes $(s, t) \in \text{rstep } R$
shows $\exists C \ \sigma \ l \ r. (l, r) \in R \wedge s = C \langle l \cdot \sigma \rangle \wedge t = C \langle r \cdot \sigma \rangle$
proof –
from *assms* **obtain** $l \ r \ p \ \sigma$ **where** $\text{step}:(s, t) \in \text{rstep-r-p-s } R \ (l, r) \ p \ \sigma$
unfolding *rstep-iff-rstep-r-p-s* **by** *best*
let $?C = \text{ctxt-of-pos-term } p \ s$
from *step* **have** $p \in \text{poss } s$ **and** $(l, r) \in R$ **and** $?C \langle l \cdot \sigma \rangle = s$ **and** $?C \langle r \cdot \sigma \rangle = t$
unfolding *rstep-r-p-s-def Let-def* **by** *auto*
then **have** $(l, r) \in R \wedge s = ?C \langle l \cdot \sigma \rangle \wedge t = ?C \langle r \cdot \sigma \rangle$ **by** *auto*
then **show** *?thesis* **by** *force*
qed

lemma *SN-Var*:
assumes *wf-trs R* **shows** *SN-on (rstep R) {Var x}*
proof (*rule ccontr*)
assume $\neg ?thesis$
then **obtain** *S* **where** [*symmetric*]: $S \ 0 = \text{Var } x$
and *chain*: *chain (rstep R) S* **by** *auto*
then **have** $(\text{Var } x, S \ (\text{Suc } 0)) \in \text{rstep } R$ **by** *force*
then **obtain** $C \ l \ r \ \sigma$ **where** $(l, r) \in R$ **and** $\text{Var } x = C \langle l \cdot \sigma \rangle$ **by** *best*
then **have** $\text{Var } x = l \cdot \sigma$ **by** (*induct C*) *simp-all*
then **obtain** *y* **where** $l = \text{Var } y$ **by** (*induct l*) *simp-all*

```

with assms and  $\langle l, r \rangle \in R$  show False unfolding wf-trs-def by blast
qed

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

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

lemma map-funs-trs-union: map-funs-trs fg (R  $\cup$  S) = map-funs-trs fg R  $\cup$  map-funs-trs fg S
unfolding map-funs-trs.simps by auto

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 –
  from step obtain C l r  $\sigma$  where s:  $s = C(l \cdot \sigma)$  and t:  $t = C(r \cdot \sigma)$  and rule:
 $(l, r) \in R$  by auto
  let ? $\sigma$  = map-funs-subst h  $\sigma$ 
  let ?h = map-funs-term h
  note funs-defs = funs-rule-def[abs-def] funs-trs-def
  from rule have lr:  $\text{funs-term } l \cup \text{funs-term } r \subseteq \text{funs-trs } R$  unfolding funs-defs
by auto
  have hl: ?h l = l
by (rule funs-term-map-funs-term-id[OF R], insert lr, auto)
  have hr: ?h r = r
by (rule funs-term-map-funs-term-id[OF R], insert lr, auto)
  show ?thesis unfolding s t
unfolding map-funs-subst-distrib map-funs-term-ctxt-distrib hl hr
by (rule rstepI[OF rule refl refl])
qed

lemma wf-trs-map-funs-trs[simp]: wf-trs (map-funs-trs f R) = wf-trs R
unfolding wf-trs-def
proof (rule iffI, intro allI impI)
  fix l r
  assume  $\forall l r. (l, r) \in \text{map-funs-trs } f R \longrightarrow (\exists f ts. l = \text{Fun } f ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l$  and  $(l, r) \in R$ 
  then show  $(\exists f ts. l = \text{Fun } f ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l$  by (cases l, force+)
next
  assume ass:  $\forall l r. (l, r) \in R \longrightarrow (\exists f ts. l = \text{Fun } f ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l$ 
  show  $\forall l r. (l, r) \in \text{map-funs-trs } f R \longrightarrow (\exists f ts. l = \text{Fun } f ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l$ 
proof (intro allI impI)
  fix l r

```

```

    assume  $(l, r) \in \text{map-funs-trs } f \ R$ 
    with ass
    show  $(\exists f \ ts. \ l = \text{Fun } f \ ts) \wedge \text{vars-term } r \subseteq \text{vars-term } l$ 
    by (cases l, force+)
  qed
qed

```

```

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 -
  have mr:  $\text{map-funs-rule } (fg \ o \ gh) = \text{map-funs-rule } fg \ o \ \text{map-funs-rule } gh$ 
  by (rule ext, auto simp: map-funs-term-comp)
  then show ?thesis
  by (auto simp: map-funs-term-comp image-comp mr)
qed

```

```

lemma map-funs-trs-mono: assumes  $R \subseteq R'$  shows  $\text{map-funs-trs } fg \ R \subseteq \text{map-funs-trs } fg \ R'$ 
using assms by auto

```

```

lemma map-funs-trs-power-mono:
  fixes  $R \ R' :: ('f, 'v)\text{trs}$  and  $fg :: 'f \Rightarrow 'v$ 
  assumes  $R \subseteq R'$  shows  $((\text{map-funs-trs } fg)^{\sim n}) \ R \subseteq ((\text{map-funs-trs } fg)^{\sim n}) \ R'$ 
using assms by (induct n, simp, auto)

```

```

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

```

```

lemma rstep-imp-map-rstep:
  assumes  $(s, t) \in \text{rstep } R$ 
  shows  $(\text{map-funs-term } fg \ s, \text{map-funs-term } fg \ t) \in \text{rstep } (\text{map-funs-trs } fg \ R)$ 
  using assms
proof (induct)
  case (IH C  $\sigma \ l \ r$ )
  then have  $(\text{map-funs-term } fg \ l, \text{map-funs-term } fg \ r) \in \text{map-funs-trs } fg \ R$  (is ( $?l, ?r$ )  $\in ?R$ )
  unfolding map-funs-trs.simps by force
  then have  $(?l, ?r) \in \text{rstep } ?R ..$ 
  then have  $(?l \cdot \text{map-funs-subst } fg \ \sigma, ?r \cdot \text{map-funs-subst } fg \ \sigma) \in \text{rstep } ?R ..$ 
  then show ?case by auto
qed

```

```

lemma rsteps-imp-map-rsteps: assumes  $(s, t) \in (\text{rstep } R)^*$ 
  shows  $(\text{map-funs-term } fg \ s, \text{map-funs-term } fg \ t) \in (\text{rstep } (\text{map-funs-trs } fg \ R))^*$ 
using assms
proof (induct, clarify)
  case (step y z)
  then have  $(\text{map-funs-term } fg \ y, \text{map-funs-term } fg \ z) \in \text{rstep } (\text{map-funs-trs } fg \ R)$ 
using rstep-imp-map-rstep
  by (auto simp: map-funs-trs.simps)

```

with *step* **show** *?case* **by** *auto*
qed

lemma *SN-map-imp-SN*:

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

shows *SN-on* (*rstep* *R*) {*t*}

proof (*rule ccontr*)

assume \neg *SN-on* (*rstep* *R*) {*t*}

from *this* **obtain** *f* **where** *cond*: $f\ 0 = t \wedge (\forall\ i. (f\ i, f\ (Suc\ i)) \in rstep\ R)$

unfolding *SN-on-def* **by** *auto*

obtain *g* **where** $g = (\lambda\ i. map-funs-term\ fg\ (f\ i))$ **by** *auto*

with *cond* **have** *cond2*: $g\ 0 = map-funs-term\ fg\ t \wedge (\forall\ i. (g\ i, g\ (Suc\ i)) \in rstep\ (map-funs-trs\ fg\ R))$

using *rstep-imp-map-rstep*[**where** *fg* = *fg*] **by** *blast*

from *SN*

have $\neg (\exists\ g. (g\ 0 = map-funs-term\ fg\ t \wedge (\forall\ i. (g\ i, g\ (Suc\ i)) \in rstep\ (map-funs-trs\ fg\ R))))$

unfolding *SN-on-def* **by** *auto*

with *cond2* **show** *False* **by** *auto*

qed

lemma *rstep-iff-map-rstep*:

assumes *inj* *fg*

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

proof

assume $(s, t) \in rstep\ R$

then **show** $(map-funs-term\ fg\ s, map-funs-term\ fg\ t) \in rstep\ (map-funs-trs\ fg\ R)$

by (*rule rstep-imp-map-rstep*)

next

assume $(map-funs-term\ fg\ s, map-funs-term\ fg\ t) \in rstep\ (map-funs-trs\ fg\ R)$

then **have** $(map-funs-term\ fg\ s, map-funs-term\ fg\ t) \in ctxt.closure(subst.closure(map-funs-trs\ fg\ R))$

by (*simp add: rstep-eq-closure*)

then **obtain** *C* *u* *v* **where** $(u, v) \in subst.closure(map-funs-trs\ fg\ R)$ **and** $C\langle u \rangle$

$= map-funs-term\ fg\ s$

and $C\langle v \rangle = map-funs-term\ fg\ t$

by (*cases rule: ctxt.closure.cases*) *force*

then **obtain** $\sigma\ w\ x$ **where** $(w, x) \in map-funs-trs\ fg\ R$ **and** $w \cdot \sigma = u$ **and** $x \cdot \sigma = v$

by (*cases rule: subst.closure.cases*) *force*

then **obtain** *l* *r* **where** $w = map-funs-term\ fg\ l$ **and** $x = map-funs-term\ fg\ r$

and $(l, r) \in R$ **by** (*auto simp: map-funs-trs.simps*)

have *ps*: $C\langle map-funs-term\ fg\ l \cdot \sigma \rangle = map-funs-term\ fg\ s$ **and** *pt*: $C\langle map-funs-term\ fg\ r \cdot \sigma \rangle = map-funs-term\ fg\ t$

unfolding $\langle w = map-funs-term\ fg\ l \rangle [symmetric]$ $\langle x = map-funs-term\ fg\ r \rangle [symmetric]$

$\langle w \cdot \sigma = u \rangle$ $\langle x \cdot \sigma = v \rangle$

$\langle C\langle u \rangle = map-funs-term\ fg\ s \rangle$ $\langle C\langle v \rangle = map-funs-term\ fg\ t \rangle$ **by** *auto*

let *?gf* = *the-inv* *fg*

let $?C = \text{map-funs-ctxt } ?gf \ C$
let $?σ = \text{map-funs-subst } ?gf \ σ$
have $gffg: ?gf \circ fg = id$ **using** *the-inv-f-f[OF assms]* **by** (*intro ext, auto*)
from ps **and** pt **have** $s = \text{map-funs-term } ?gf \ (C \langle \text{map-funs-term } fg \ l \cdot σ \rangle)$
and $t = \text{map-funs-term } ?gf \ (C \langle \text{map-funs-term } fg \ r \cdot σ \rangle)$ **by** (*auto simp:*
map-funs-term-comp gffg)
then have $s: s = ?C \langle \text{map-funs-term } ?gf \ (\text{map-funs-term } fg \ l \cdot σ) \rangle$
and $t: t = ?C \langle \text{map-funs-term } ?gf \ (\text{map-funs-term } fg \ r \cdot σ) \rangle$ **using** *map-funs-term-ctxt-distrib*
by *auto*
from s **have** $s = ?C \langle l \cdot ?σ \rangle$ **by** (*simp add: map-funs-term-comp gffg*)
from t **have** $t = ?C \langle r \cdot ?σ \rangle$ **by** (*simp add: map-funs-term-comp gffg*)
from $\langle l, r \rangle \in R$ **have** $(l \cdot ?σ, r \cdot ?σ) \in \text{subst.closure } R$ **by** *blast*
then have $(?C \langle l \cdot ?σ \rangle, ?C \langle r \cdot ?σ \rangle) \in \text{ctxt.closure}(\text{subst.closure } R)$ **by** *blast*
then show $(s, t) \in \text{rstep } R$ **unfolding** $\langle s = ?C \langle l \cdot ?σ \rangle \rangle \langle t = ?C \langle r \cdot ?σ \rangle \rangle$ **by** (*simp*
add: rstep-eq-closure)
qed

lemma *rstep-map-funs-trs-power-mono*:
fixes $R \ R' :: ('f, 'v)\text{trs}$ **and** $fg :: 'f \Rightarrow 'v$
assumes *subset*: $R \subseteq R'$ **shows** $\text{rstep } (((\text{map-funs-trs } fg) \sim^n) R) \subseteq \text{rstep } (((\text{map-funs-trs } fg) \sim^n) R')$
by (*rule rstep-mono, rule map-funs-trs-power-mono, rule subset*)

lemma *subsetI3*: $(\bigwedge x \ y \ z. (x, y, z) \in A \implies (x, y, z) \in B) \implies A \subseteq B$ **by** *auto*

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

proof

show $?P \subseteq ?Q$

proof

fix x **assume** $x \in ?P$

then obtain a **where** $a \in P$ **and** $x \in \{(x, y, z). x = \text{fst } a \wedge y = \text{snd } a \wedge Q \ a \ z\}$

by *auto*

then obtain b **where** $Q \ (\text{fst } x, \text{fst } (\text{snd } x)) \ (\text{snd } (\text{snd } x))$ **and** $(\text{fst } x, \text{fst } (\text{snd } x)) = a$ **and** $\text{snd } (\text{snd } x) = b$ **by** *force*

from $\langle a \in P \rangle$ **have** $(\text{fst } a, \text{snd } a) \in P$ **unfolding** *split-def* **by** *simp*

from $\langle Q \ (\text{fst } x, \text{fst } (\text{snd } x)) \ (\text{snd } (\text{snd } x)) \rangle$ **have** $Q \ a \ b$ **unfolding** $\langle (\text{fst } x, \text{fst } (\text{snd } x)) = a \rangle \langle \text{snd } (\text{snd } x) = b \rangle$.

from $\langle (\text{fst } a, \text{snd } a) \in P \rangle$ **and** $\langle Q \ a \ b \rangle$ **show** $x \in ?Q$ **unfolding** *split-def* $\langle (\text{fst } x, \text{fst } (\text{snd } x)) = a \rangle [\text{symmetric}] \langle \text{snd } (\text{snd } x) = b \rangle [\text{symmetric}]$ **by** *simp*

qed

next

show $?Q \subseteq ?P$

proof (*rule subsetI3*)

fix $x \ y \ z$ **assume** $(x, y, z) \in ?Q$

then have $(x, y) \in P$ **and** $Q \ (x, y) \ z$ **by** *auto*

then have $x = \text{fst}(x, y) \wedge y = \text{snd}(x, y) \wedge Q \ (x, y) \ z$ **by** *auto*

then have $(x, y, z) \in \{(x, y, z). x = \text{fst}(x, y) \wedge y = \text{snd}(x, y) \wedge Q \ (x, y) \ z\}$ **by** *auto*

then have $\exists p \in P. (x, y, z) \in \{(x, y, z). x = \text{fst } p \wedge y = \text{snd } p \wedge Q \ p \ z\}$ **using**

$\langle (x,y) \in P \rangle$ **by** *blast*
 then **show** $(x,y,z) \in ?P$ **unfolding** *UN-iff[symmetric]* **by** *simp*
qed
qed

lemma *finite-imp-finite-DP-on'*:
 assumes *finite R*
 shows *finite* $\{(l, r, u)\}$.
 $\exists h \text{ us. } u = \text{Fun } h \text{ us} \wedge (l, r) \in R \wedge r \supseteq u \wedge (h, \text{length us}) \in F \wedge \neg (l \triangleright u)\}$
proof –
 have $\bigwedge l r. (l, r) \in R \implies \text{finite } \{u. r \supseteq u\}$
proof –
 fix $l r$
 assume $(l, r) \in R$
 show *finite* $\{u. r \supseteq u\}$ **by** (*rule finite-subterms*)
qed
 with $\langle \text{finite } R \rangle$ **have** *finite* $(\bigcup (l, r) \in R. \{u. r \supseteq u\})$ **by** *auto*
 have *finite* $(\bigcup lr \in R. \{(l, r, u). l = \text{fst } lr \wedge r = \text{snd } lr \wedge \text{snd } lr \supseteq u\})$
proof (*rule finite-UN-I*)
 show *finite R* **by** (*rule* $\langle \text{finite } R \rangle$)
next
 fix lr
 assume $lr \in R$
 have *finite* $\{u. \text{snd } lr \supseteq u\}$ **by** (*rule finite-subterms*)
 then **show** *finite* $\{(l,r,u). l = \text{fst } lr \wedge r = \text{snd } lr \wedge \text{snd } lr \supseteq u\}$ **by** *auto*
qed
 then **have** *finite* $\{(l,r,u). (l,r) \in R \wedge r \supseteq u\}$ **unfolding** *aux* **by** *auto*
 have $\{(l,r,u). (l,r) \in R \wedge r \supseteq u\} \supseteq \{(l,r,u). (\exists h \text{ us. } u = \text{Fun } h \text{ us} \wedge (l,r) \in R \wedge r \supseteq u \wedge (h, \text{length us}) \in F \wedge \neg (l \triangleright u))\}$ **by** *auto*
 with $\langle \text{finite } \{(l,r,u). (l,r) \in R \wedge r \supseteq u\} \rangle$ **show** *?thesis* **using** *finite-subset* **by** *fast*
qed

lemma *card-image-le'*:
 assumes *finite S*
 shows *card* $(\bigcup y \in S. \{x. x = f y\}) \leq \text{card } S$
proof –
 have $A: (\bigcup y \in S. \{x. x = f y\}) = f' S$ **by** *auto*
 from *assms* **show** *?thesis* **unfolding** *A* **using** *card-image-le* **by** *auto*
qed

lemma *subteq-of-map-imp-map*: *map-funs-term* $g \ s \supseteq t \implies \exists u. t = \text{map-funs-term } g \ u$
proof (*induct s arbitrary: t*)
 case (*Var x*)
 then **have** *map-funs-term* $g \ (\text{Var } x) \triangleright t \vee \text{map-funs-term } g \ (\text{Var } x) = t$ **by** *auto*
 then **show** *?case*
proof
 assume *map-funs-term* $g \ (\text{Var } x) \triangleright t$ then **show** *?thesis* **by** (*cases rule: supt.cases*)

```

auto
next
  assume map-funs-term g (Var x) = t then show ?thesis by best
qed
next
case (Fun f ss)
then have map-funs-term g (Fun f ss)  $\triangleright$  t  $\vee$  map-funs-term g (Fun f ss) = t by
auto
then show ?case
proof
  assume map-funs-term g (Fun f ss)  $\triangleright$  t
  then show ?case using Fun by (cases rule: supt.cases) (auto simp: supt-supteq-conv)
next
  assume map-funs-term g (Fun f ss) = t then show ?thesis by best
qed
qed

lemma map-funs-term-inj:
  assumes inj (fg :: ('f  $\Rightarrow$  'g))
  shows inj (map-funs-term fg)
proof -
  {
    fix s t :: ('f, 'v)term
    assume map-funs-term fg s = map-funs-term fg t
    then have s = t
    proof (induct s arbitrary: t)
      case (Var x) with assms show ?case by (induct t) auto
    next
      case (Fun f ss) then show ?case
      proof (induct t)
        case (Var y) then show ?case by auto
      next
        case (Fun g ts)
        then have A: map (map-funs-term fg) ss = map (map-funs-term fg) ts by
simp
        then have len-eq: length ss = length ts by (rule map-eq-imp-length-eq)
        from A have !!i. i < length ss  $\implies$  (map (map-funs-term fg) ss)!i = (map
(map-funs-term fg) ts)!i by auto
        with len-eq have eq: !!i. i < length ss  $\implies$  map-funs-term fg (ss!i) =
map-funs-term fg (ts!i) using nth-map by auto
        have in-set: !!i. i < length ss  $\implies$  (ss!i)  $\in$  set ss by auto
        from Fun have  $\forall i < \text{length } ss. (ss!i) = (ts!i)$  using in-set eq by auto
        with len-eq have ss = ts using nth-equalityI[where xs = ss and ys = ts]
by simp
        have f = g using Fun <inj fg> unfolding inj-on-def by auto
        with <ss = ts> show ?case by simp
      qed
    qed
  }
}

```

then show *?thesis unfolding inj-on-def* by auto
qed

lemma *rsteps-closed-ctxt*:

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

proof –

from *assms* obtain n where $(s, t) \in (rstep\ R)^{\sim n}$
using *rtrancl-is-UN-relpow* by auto

then show *?thesis*

proof (induct n arbitrary: s)

case 0 then show *?case* by auto

next

case (Suc n)

from *relpow-Suc-D2[OF $\langle s, t \rangle \in (rstep\ R)^{\sim Suc\ n}$]* obtain u

where $(s, u) \in (rstep\ R)$ and $(u, t) \in (rstep\ R)^{\sim n}$ by auto

from $\langle s, u \rangle \in (rstep\ R)$ have $(C\langle s \rangle, C\langle u \rangle) \in (rstep\ R)$..

from *Suc* and $\langle u, t \rangle \in (rstep\ R)^{\sim n}$ have $(C\langle u \rangle, C\langle t \rangle) \in (rstep\ R)^*$ by *simp*

with $\langle C\langle s \rangle, C\langle u \rangle \rangle \in (rstep\ R)$ show *?case* by auto

qed

qed

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

proof (induct xs)

case Nil then show *?case* by *simp*

next

case (Cons $x\ xs$) show *?case*

proof (cases xs)

assume $xs = []$ with *Cons* show *?thesis* by *simp*

next

fix $x'\ xs'$ assume $xs = x' \# xs'$

then have $xs \neq []$ by *simp*

with *Cons* obtain $ys\ y$ where $xs = ys@[y]$ by auto

then have $x \# xs = x \# (ys@[y])$ by *simp*

then have $x \# xs = (x \# ys)@[y]$ by *simp*

then show *?thesis* by auto

qed

qed

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

using *not-Nil-imp-last* by *blast*

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

proof

fix $s\ t\ C$

assume *st*: $(s, t) \in r$

show $(C\langle s \rangle, C\langle t \rangle) \in r$

```

proof (induct C)
  case (More f bef C aft)
  from one[OF More] show ?case by auto
qed (insert st, auto)
qed

```

```

lemma ctxt-closed-nrrstep [intro]: ctxt.closed (nrrstep R)
proof (rule one-imp-ctxt-closed)
  fix f bef s t aft
  assume (s,t) ∈ nrrstep R
  from this[unfolded nrrstep-def] obtain l r C σ
  where lr: (l,r) ∈ R and C: C ≠ □
  and s: s = C⟨l.σ⟩ and t: t = C⟨r · σ⟩ by auto
  show (Fun f (bef @ s # aft), Fun f (bef @ t # aft)) ∈ nrrstep R
  proof (rule nrrstepI[OF lr])
    show More f bef C aft ≠ □ by simp
  qed (insert s t, auto)
qed

```

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

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

```

lemma all-ctxt-closedD: all-ctxt-closed F r ⇒ (f,length ss) ∈ F ⇒ length ts =
length ss
⇒ [⋀ i. i < length ts ⇒ (ts ! i, ss ! i) ∈ r ]
⇒ [⋀ i. i < length ts ⇒ funas-term (ts ! i) ⊆ F ]
⇒ [⋀ i. i < length ts ⇒ funas-term (ss ! i) ⊆ F ]
⇒ (Fun f ts, Fun f ss) ∈ r
unfolding all-ctxt-closed-def by auto

```

```

lemma all-ctxt-closed-sig-reflE: assumes all: all-ctxt-closed F r
shows funas-term t ⊆ F ⇒ (t,t) ∈ r
proof (induct t)
  case (Var x)
  from all[unfolded all-ctxt-closed-def] show ?case by auto
next
  case (Fun f ts)
  show ?case
  by (rule all-ctxt-closedD[OF all - Fun(1)], insert Fun(2), force+)
qed

```

```

lemma all-ctxt-closed-reflE: assumes all: all-ctxt-closed UNIV r
shows (t,t) ∈ r
by (rule all-ctxt-closed-sig-reflE[OF all], auto)

```

```

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

```

S
shows *all-ctxt-closed UNIV* ($R \ O \ S$)
unfolding *all-ctxt-closed-def*
proof (*intro allI impI conjI*)
show ($\text{Var } x, \text{Var } x$) $\in R \ O \ S$ **for** x **using** *assms unfolding all-ctxt-closed-def*
by *auto*
fix $f \ ts \ ss$
assume $\text{len}: \text{length } ts = \text{length } ss$
and $\text{steps}: \forall i < \text{length } ts. (ts ! i, ss ! i) \in R \ O \ S$
hence $\forall i. \exists us. i < \text{length } ts \longrightarrow (ts ! i, us) \in R \wedge (us, ss ! i) \in S$ **by** *blast*
from *choice[OF this]* **obtain** us **where** $\text{steps}: \bigwedge i. i < \text{length } ts \implies (ts ! i, us \ i)$
 $\in R \wedge (us \ i, ss ! i) \in S$
by *blast*
let $?us = \text{map } us \ [0..<\text{length } ss]$
from *all-ctxt-closedD[OF assms(2)] steps len* **have** $us: (\text{Fun } f \ ?us, \text{Fun } f \ ss) \in$
 S **by** *auto*
from *all-ctxt-closedD[OF assms(1)] steps len* **have** $tu: (\text{Fun } f \ ts, \text{Fun } f \ ?us) \in R$
by *force*
from $tu \ us$
show $(\text{Fun } f \ ts, \text{Fun } f \ ss) \in R \ O \ S$ **by** *auto*
qed

lemma *all-ctxt-closed-relpow*:
assumes $\text{acc}: \text{all-ctxt-closed UNIV } Q$
shows *all-ctxt-closed UNIV* ($Q \rightsquigarrow^n$)
proof (*induct n*)
case 0
thus $?case$ **by** (*auto simp: all-ctxt-closed-def nth-equalityI*)
next
case ($\text{Suc } n$)
from *all-ctxt-closed-relcomp[OF this acc]*
show $?case$ **by** *simp*
qed

lemma *all-ctxt-closed-subst-step-sig*:
fixes $r :: ('f, 'v) \text{trs}$ **and** $t :: ('f, 'v) \text{term}$
assumes $\text{all}: \text{all-ctxt-closed } F \ r$
and $\text{sig}: \text{funas-term } t \subseteq F$
and $\text{steps}: \bigwedge x. x \in \text{vars-term } t \implies (\sigma \ x, \tau \ x) \in r$
and $\text{sig-subst}: \bigwedge x. x \in \text{vars-term } t \implies \text{funas-term } (\sigma \ x) \cup \text{funas-term } (\tau \ x)$
 $\subseteq F$
shows $(t \cdot \sigma, t \cdot \tau) \in r$
using *sig steps sig-subst*
proof (*induct t*)
case ($\text{Var } x$)
then show $?case$ **by** *auto*
next
case ($\text{Fun } f \ ts$)

```

{
  fix t
  assume t: t ∈ set ts
  with Fun(2-3) have funas-term t ⊆ F ∧ x. x ∈ vars-term t ⇒ (σ x, τ x)
∈ r by auto
  from Fun(1)[OF t this Fun(4)] t have step: (t · σ, t · τ) ∈ r by auto
  from Fun(4) t have ∧ x. x ∈ vars-term t ⇒ funas-term (σ x) ∪ funas-term
(τ x) ⊆ F by auto
  with ⟨funas-term t ⊆ F⟩ have funas-term (t · σ) ∪ funas-term (t · τ) ⊆ F
unfolding funas-term-subst by auto
  note step this
}
then have steps: ∧ i. i < length ts ⇒ (ts ! i · σ, ts ! i · τ) ∈ r ∧ funas-term
(ts ! i · σ) ∪ funas-term (ts ! i · τ) ⊆ F unfolding set-conv-nth by auto
with all-ctxt-closedD[OF all, of f map (λ t. t · τ) ts map (λ t. t · σ) ts] Fun(2)
show ?case by auto
qed

```

```

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: ∧ x. x ∈ vars-term t ⇒ (σ x, τ x) ∈ r
  shows (t · σ, t · τ) ∈ r
  by (rule all-ctxt-closed-subst-step-sig[OF all - steps], auto)

```

```

lemma all-ctxt-closed-ctxtE: assumes all: all-ctxt-closed F R
  and Fs: funas-term s ⊆ F
  and Ft: funas-term t ⊆ F
  and step: (s, t) ∈ R
  shows funas-ctxt C ⊆ F ⇒ (C ⟨ s ⟩, C ⟨ t ⟩) ∈ R
proof(induct C)
  case Hole
  from step show ?case by auto
next
  case (More f bef C aft)
  let ?n = length bef
  let ?m = Suc (?n + length aft)
  show ?case unfolding ctxt-apply-term.simps
  proof (rule all-ctxt-closedD[OF all])
    fix i
    let ?t = λ s. (bef @ C ⟨ s ⟩ # aft) ! i
    assume i < length (bef @ C ⟨ s ⟩ # aft)
    then have i: i < ?m by auto
    then have mem: ∧ s. ?t s ∈ set (bef @ C ⟨ s ⟩ # aft) unfolding set-conv-nth
  by auto
  from mem[of s] More Fs show funas-term (?t s) ⊆ F by auto
  from mem[of t] More Ft show funas-term (?t t) ⊆ F by auto
  from More have step: (C ⟨ s ⟩, C ⟨ t ⟩) ∈ R by auto
  {

```

```

    fix s
    assume s ∈ set bef ∪ set aft
    with More have funas-term s ⊆ F by auto
    from all-ctxt-closed-sig-reflE[OF all this] have (s,s) ∈ R by auto
  } note steps = this
  show (?t s, ?t t) ∈ R
  proof (cases i = ?n)
    case True
    then show ?thesis using step by auto
  next
    case False
    show ?thesis
    proof (cases i < ?n)
      case True
      then show ?thesis unfolding append-Cons-nth-left[OF True] using steps
    by auto
  next
    case False
    with ⟨i ≠ ?n⟩ i have ∃ k. k < length aft ∧ i = Suc ?n + k by presburger
    then obtain k where k: k < length aft and i: i = Suc ?n + k by auto
    from k show ?thesis using steps unfolding i by (auto simp: nth-append)
  qed
  qed
  qed (insert More, auto)
qed

lemma trans-ctxt-sig-imp-all-ctxt-closed: assumes tran: trans r
  and refl: ⋀ t. funas-term t ⊆ F ⟹ (t,t) ∈ r
  and ctxt: ⋀ C s t. funas-ctxt C ⊆ F ⟹ funas-term s ⊆ F ⟹ funas-term t ⊆
F ⟹ (s,t) ∈ r ⟹ (C ⟨ s ⟩, C ⟨ t ⟩) ∈ r
  shows all-ctxt-closed F r
unfolding all-ctxt-closed-def
proof (intro conjI, intro allI impI)
  fix f ts ss
  assume f: (f,length ss) ∈ F and
    l: length ts = length ss and
    steps: ∀ i < length ts. (ts ! i, ss ! i) ∈ r and
    sig: ∀ i < length ts. funas-term (ts ! i) ∪ funas-term (ss ! i) ⊆ F
  from sig have sig-ts: ⋀ t. t ∈ set ts ⟹ funas-term t ⊆ F unfolding set-conv-nth
by auto
  let ?p = λ ss. (Fun f ts, Fun f ss) ∈ r ∧ funas-term (Fun f ss) ⊆ F
  let ?r = λ xsi ysi. (xsi, ysi) ∈ r ∧ funas-term ysi ⊆ F
  have init: ?p ts by (rule conjI[OF refl], insert f sig-ts l, auto)
  have ?p ss
  proof (rule parallel-list-update[where p = ?p and r = ?r, OF - HOL.refl init
l[symmetric]])
    fix xs i y
    assume len: length xs = length ts
    and i: i < length ts

```

```

    and r: ?r (xs ! i) y
    and p: ?p xs
    let ?C = More f (take i xs) Hole (drop (Suc i) xs)
    have id1: Fun f xs = ?C < xs ! i> using id-take-nth-drop[OF i[folded len]] by
simp
    have id2: Fun f (xs[i := y]) = ?C < y> using upd-conv-take-nth-drop[OF
i[folded len]] by simp
    from p[unfolded id1] have C: funas-ctxt ?C  $\subseteq$  F and xi: funas-term (xs ! i)
 $\subseteq$  F by auto
    from r have funas-term y  $\subseteq$  F (xs ! i, y)  $\in$  r by auto
    with ctxt[OF C xi this] C have r: (Fun f xs, Fun f (xs[i := y]))  $\in$  r
    and f: funas-term (Fun f (xs[i := y]))  $\subseteq$  F unfolding id1 id2 by auto
    from p r tran have (Fun f ts, Fun f (xs[i := y]))  $\in$  r unfolding trans-def by
auto
    with f
    show ?p (xs[i := y]) by auto
  qed (insert sig steps, auto)
  then show (Fun f ts, Fun f ss)  $\in$  r ..
qed (insert refl, auto)

```

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*
 by (rule *trans-ctxt-sig-imp-all-ctxt-closed*[OF *tran* - *ctxt.closedD*[OF *ctxt*]], insert
refl[unfolded *refl-on-def*], auto)

lemma *all-ctxt-closed-rsteps*[intro]: *all-ctxt-closed* *F* ((*rstep* *r*)*)
 by (blast intro: *trans-ctxt-imp-all-ctxt-closed* *trans-rtrancl* *refl-rtrancl*)

lemma *subst-rsteps-imp-rsteps*:
 fixes $\sigma :: ('f, 'v)$ *subst*
 assumes $\forall x \in \text{vars-term } t. (\sigma x, \tau x) \in (\text{rstep } R)^*$
 shows $(t \cdot \sigma, t \cdot \tau) \in (\text{rstep } R)^*$
 by (rule *all-ctxt-closed-subst-step*)
 (insert *assms*, auto)

lemma *rtrancl-trancl-into-trancl*:
 assumes *len*: *length* *ts* = *length* *ss*
 and *steps*: $\forall i < \text{length } ts. (ts ! i, ss ! i) \in R^*$
 and *i*: *i* < *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 –
 from *ctxt* have *ctxt-rt*: *ctxt.closed* (R^*) by blast
 from *ctxt* have *ctxt-t*: *ctxt.closed* (R^+) by blast
 from *id-take-nth-drop*[OF *i*] have *ts*: *ts* = *take* *i* *ts* @ *ts* ! *i* # *drop* (*Suc* *i*) *ts* (is
 - = ?*ts*) by auto

```

from id-take-nth-drop[OF i[simplified len]] have ss: ss = take i ss @ ss ! i #
drop (Suc i) ss (is - = ?ss) by auto
let ?mid = take i ss @ ts ! i # drop (Suc i) ss
from trans-ctxt-imp-all-ctxt-closed[OF trans-rtrancl refl-rtrancl ctxt-rt] have all:
all-ctxt-closed UNIV (R*) .
from ctxt-closed-one[OF ctxt-t step] have (Fun f ?mid, Fun f ?ss) ∈  $R^+$  .
then have part1: (Fun f ?mid, Fun f ss) ∈  $R^+$  unfolding ss[symmetric] .
from ts have lents: length ts = length ?ts by simp
have (Fun f ts, Fun f ?mid) ∈  $R^*$ 
proof (rule all-ctxt-closedD[OF all])
  fix j
  assume jts: j < length ts
  from i len have i: i < length ss by auto
  show (ts ! j, ?mid ! j) ∈  $R^*$ 
  proof (cases j < i)
    case True
    with i have j: j < length ss by auto
    with True have id: ?mid ! j = ss ! j by (simp add: nth-append)
    from steps len j have (ts ! j, ss ! j) ∈  $R^*$  by auto
    then show ?thesis using id by simp
  next
  case False
  show ?thesis
  proof (cases j = i)
    case True
    then have ?mid ! j = ts ! j using i by (simp add: nth-append)
    then show ?thesis by simp
  next
  case False
  from i have min: min (length ss) i = i by simp
  from False <⊃ j < i have j > i by arith
  then obtain k where k: j - i = Suc k by (cases j - i, auto)
  then have j: j = Suc (i+k) by auto
  with jts len have ss: Suc i + k ≤ length ss and jlen: j < length ts by auto
  then have ?mid ! j = ss ! j using j i by (simp add: nth-append min <⊃ j
  < i) nth-drop[OF ss])
  with steps jlen show ?thesis by auto
  qed
qed
qed (insert lents[symmetric] len, auto)
with part1 show ?thesis by auto
qed

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 (set (ctxt-to-term-list C))
proof –
  {

```

```

fix u
assume u ∈ set (ctxt-to-term-list C)
from ctxt-to-term-list-supt[OF this, of t] have C⟨t⟩ ⊇ u
  by (rule supt-imp-supteq)
from ctxt-closed-SN-on-subst[OF ctxt, OF SN this]
have SN-on r {u} by auto
}
then show ?thesis
  unfolding SN-on-def by auto
qed

```

lemma *rstep-subset*: $ctxt.closed\ R' \implies subst.closed\ R' \implies R \subseteq R' \implies rstep\ R \subseteq R'$ by *fast*

lemma *tranc1-rstep-ctxt*:
 $(s, t) \in (rstep\ R)^+ \implies (C\langle s \rangle, C\langle t \rangle) \in (rstep\ R)^+$
 by (rule ctxt.closedD, blast)

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 $(Fun\ f\ ss, Fun\ f\ ts) \in R^*$
proof –
 let $?tss = \lambda i. take\ i\ ts\ @\ drop\ i\ ss$
 {
 fix $bef :: ('f, 'v)\ term\ list$ and $s\ t$ and $aft :: ('f, 'v)\ term\ list$
 assume $(s, t) \in (r\ (length\ bef))^*$ and $len: length\ ts = Suc\ (length\ bef + length\ aft)$
 then have $(Fun\ f\ (bef\ @\ s\ \# \ aft), Fun\ f\ (bef\ @\ t\ \# \ aft)) \in R^*$
proof (induct)
 case (step t u)
 from step(3)[OF len] ctxt[OF step(2) len] show ?case by auto
 qed simp
 }
 note one = this
 have $a: \forall i \leq length\ ts. (Fun\ f\ ss, Fun\ f\ (?tss\ i)) \in R^*$
proof (intro allI impI)
 fix i assume $i \leq length\ ts$ then show $(Fun\ f\ ss, Fun\ f\ (?tss\ i)) \in R^*$
proof (induct i)
 case 0
 then show ?case by simp
next
 case (Suc i)
 then have IH: $(Fun\ f\ ss, Fun\ f\ (?tss\ i)) \in R^*$
 and $i: i < length\ ts$ by auto
 have $si: ?tss\ (Suc\ i) = take\ i\ ts\ @\ ts\ !\ i\ \# \ drop\ (Suc\ i)\ ss$

```

    unfolding take-Suc-conv-app-nth[OF i] by simp
    have ii: ?ts i = take i ts @ ss ! i # drop (Suc i) ss
    unfolding Cons-nth-drop-Suc[OF i[unfolded len[symmetric]]] ..
    from i have i': length (take i ts) < length ts and len': length (take i ts) = i
  by auto
    from len i have len'': length ts = Suc (length (take i ts) + length (drop (Suc
i) ss)) by simp
    from one[OF args[OF i'] len''] IH
    show ?case unfolding si ii len' by auto
  qed
qed
from this[THEN spec, THEN mp[OF - le-refl]]
show ?thesis using len by auto
qed

```

```

lemma args-steps-imp-steps:
  assumes ctxt: ctxt.closed R
    and len: length ss = length ts and args:  $\forall i < \text{length } ss. (ss!i, ts!i) \in R^*$ 
  shows (Fun f ss, Fun f ts)  $\in R^*$ 
proof (rule args-steps-imp-steps-gen[OF - len])
  fix i
    assume i < length ts then show (ss ! i, ts ! i)  $\in R^*$  using args len by auto
qed (insert ctxt-closed-one[OF ctxt], auto)

```

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

```

lemma replace-at-subst-steps:
  fixes  $\sigma \tau :: ('f, 'v) \text{subst}$ 
  assumes acc: all-ctxt-closed UNIV r
    and refl: refl r
    and *:  $\bigwedge x. (\sigma x, \tau x) \in r$ 
    and p  $\in \text{poss } t$ 
    and t  $\mid\!-\! p = \text{Var } x$ 
  shows (replace-at (t  $\cdot$   $\sigma$ ) p ( $\tau x$ ), t  $\cdot$   $\tau$ )  $\in r$ 
  using assms(4-)
proof (induction t arbitrary: p)
  case (Var x)
    then show ?case using refl by (simp add: refl-on-def)
next
  case (Fun f ts)
    then obtain i q where [simp]: p = i # q and i: i < length ts
      and q: q  $\in \text{poss } (ts ! i)$  and [simp]: ts ! i  $\mid\!-\! q = \text{Var } x$  by (cases p) auto
    let ?C = ctxt-of-pos-term q (ts ! i  $\cdot$   $\sigma$ )
    let ?ts = map ( $\lambda t. t \cdot \tau$ ) ts
    let ?ss = take i (map ( $\lambda t. t \cdot \sigma$ ) ts) @ ?C( $\tau x$ ) # drop (Suc i) (map ( $\lambda t. t \cdot \sigma$ )
ts)
    have  $\forall j < \text{length } ts. (?ss ! j, ?ts ! j) \in r$ 
  proof (intro allI impI)
    fix j

```

```

    assume  $j: j < \text{length } ts$ 
    moreover
    { assume  $[simp]: j = i$ 
      have  $?ss ! j = ?C \langle \tau \ x \rangle$  using  $i$  by (simp add: nth-append-take)
      with  $Fun.IH$  [of  $ts ! i \ q$ ]
      have  $(?ss ! j, ?ts ! j) \in r$  using  $q$  and  $i$  by simp }
    moreover
    { assume  $j < i$ 
      with  $i$  have  $?ss ! j = ts ! j \cdot \sigma$ 
        and  $?ts ! j = ts ! j \cdot \tau$  by (simp-all add: nth-append-take-is-nth-conv)
      then have  $(?ss ! j, ?ts ! j) \in r$  by (auto simp: * all-ctxt-closed-subst-step [OF
acc]) }
    moreover
    { assume  $j > i$ 
      with  $i$  and  $j$  have  $?ss ! j = ts ! j \cdot \sigma$ 
        and  $?ts ! j = ts ! j \cdot \tau$  by (simp-all add: nth-append-drop-is-nth-conv)
      then have  $(?ss ! j, ?ts ! j) \in r$  by (auto simp: * all-ctxt-closed-subst-step [OF
acc]) }
    ultimately show  $(?ss ! j, ?ts ! j) \in r$  by arith
  qed
  moreover have  $i < \text{length } ts$  by fact
  ultimately show  $?case$ 
    by (auto intro: all-ctxt-closedD [OF acc])
qed

```

```

lemma replace-at-subst-rsteps:
  fixes  $\sigma \ \tau :: ('f, 'v) \text{subst}$ 
  assumes *:  $\bigwedge x. (\sigma \ x, \tau \ x) \in (\text{rstep } R)^*$ 
  and  $p \in \text{poss } t$ 
  and  $t \mid - \ p = \text{Var } x$ 
  shows  $(\text{replace-at } (t \cdot \sigma) \ p \ (\tau \ x), t \cdot \tau) \in (\text{rstep } R)^*$ 
  by (intro replace-at-subst-rsteps[OF - - assms], auto simp: refl-on-def)

```

```

lemma substs-rsteps:
  assumes  $\bigwedge x. (\sigma \ x, \tau \ x) \in (\text{rstep } R)^*$ 
  shows  $(t \cdot \sigma, t \cdot \tau) \in (\text{rstep } R)^*$ 
proof (induct  $t$ )
  case (Var  $y$ )
  show  $?case$  using assms by simp-all
next
  case (Fun  $f \ ts$ )
  then have  $\forall i < \text{length } (map (\lambda t. t \cdot \sigma) \ ts).$ 
     $(map (\lambda t. t \cdot \sigma) \ ts ! i, map (\lambda t. t \cdot \tau) \ ts ! i) \in (\text{rstep } R)^*$  by auto
  from args-rsteps-imp-rsteps [OF - this] show  $?case$  by simp
qed

```

```

lemma nrrstep-Fun-imp-arg-rstep:
  fixes  $ss :: ('f, 'v) \text{term list}$ 
  assumes  $(Fun \ f \ ss, Fun \ f \ ts) \in \text{nrrstep } R$  (is  $(?s, ?t) \in \text{nrrstep } R$ )

```

shows $\exists C \ i. \ i < \text{length } ss \wedge (ss!i, ts!i) \in \text{rstep } R \wedge C\langle ss!i \rangle = \text{Fun } f \ ss \wedge C\langle ts!i \rangle = \text{Fun } f \ ts$
proof –
from $\langle ?s, ?t \rangle \in \text{nrrstep } R$
obtain $l \ r \ j \ ps \ \sigma$ **where** A : **let** $C = \text{ctxt-of-pos-term } (j\#ps) \ ?s$ **in** $(j\#ps) \in \text{poss } ?s \wedge (l, r) \in R \wedge C\langle l.\sigma \rangle = ?s \wedge C\langle r.\sigma \rangle = ?t$ **unfolding** $\text{nrrstep-def rstep-r-p-s-def}$
by force
let $?C = \text{ctxt-of-pos-term } (j\#ps) \ ?s$
from A **have** $(j\#ps) \in \text{poss } ?s$ **and** $(l, r) \in R$ **and** $?C\langle l.\sigma \rangle = ?s$ **and** $?C\langle r.\sigma \rangle = ?t$ **using** Let-def **by auto**
have C : $?C = \text{More } f \ (\text{take } j \ ss) \ (\text{ctxt-of-pos-term } ps \ (ss!j)) \ (\text{drop } (\text{Suc } j) \ ss)$ **(is** $?C = \text{More } f \ ?ss1 \ ?D \ ?ss2)$ **by auto**
from $\langle ?C\langle l.\sigma \rangle = ?s \rangle$ **have** $?D\langle l.\sigma \rangle = (ss!j)$ **by** $(\text{auto simp: take-drop-imp-nth})$
from $\langle (l, r) \in R \rangle$ **have** $(l.\sigma, r.\sigma) \in (\text{subst.closure } R)$ **by auto**
then have $(?D\langle l.\sigma \rangle, ?D\langle r.\sigma \rangle) \in (\text{ctxt.closure}(\text{subst.closure } R))$ **and** $(?C\langle l.\sigma \rangle, ?C\langle r.\sigma \rangle) \in (\text{ctxt.closure}(\text{subst.closure } R))$ **by** $(\text{auto simp only: ctxt.closure.intros})$
then have $D\text{-rstep: } (?D\langle l.\sigma \rangle, ?D\langle r.\sigma \rangle) \in \text{rstep } R$ **and** $(?C\langle l.\sigma \rangle, ?C\langle r.\sigma \rangle) \in \text{rstep } R$
by $(\text{auto simp: rstep-eq-closure})$
from $\langle ?C\langle r.\sigma \rangle = ?t \rangle$ **and** C **have** $?t = \text{Fun } f \ (\text{take } j \ ss \ @ \ ?D\langle r.\sigma \rangle \ # \ \text{drop } (\text{Suc } j) \ ss)$ **by auto**
then have ts : $ts = (\text{take } j \ ss \ @ \ ?D\langle r.\sigma \rangle \ # \ \text{drop } (\text{Suc } j) \ ss)$ **by auto**
from $\langle j\#ps \in \text{poss } ?s \rangle$ **have** $r0$: $j < \text{length } ss$ **by simp**
then have $(\text{take } j \ ss \ @ \ ?D\langle r.\sigma \rangle \ # \ \text{drop } (\text{Suc } j) \ ss) ! j = ?D\langle r.\sigma \rangle$ **by** $(\text{auto simp: nth-append-take})$
with ts **have** $ts!j = ?D\langle r.\sigma \rangle$ **by auto**
let $?C' = \text{More } f \ (\text{take } j \ ss) \ \square \ (\text{drop } (\text{Suc } j) \ ss)$
from $D\text{-rstep}$ **have** $r1$: $(ss!j, ts!j) \in \text{rstep } R$ **unfolding** $\langle ts!j = ?D\langle r.\sigma \rangle \rangle \ \langle ?D\langle l.\sigma \rangle = ss!j \rangle$ **by simp**
have $?s = ?C\langle l.\sigma \rangle$ **unfolding** $\langle ?C\langle l.\sigma \rangle = ?s \rangle$ **by simp**
also have $\dots = ?C'\langle ?D\langle l.\sigma \rangle \rangle$ **unfolding** C **by simp**
finally have $r2$: $?C'\langle ss!j \rangle = ?s$ **unfolding** $\langle ?D\langle l.\sigma \rangle = ss!j \rangle$ **by simp**
have $?t = ?C\langle r.\sigma \rangle$ **unfolding** $\langle ?C\langle r.\sigma \rangle = ?t \rangle$ **by simp**
also have $\dots = ?C'\langle ?D\langle r.\sigma \rangle \rangle$ **unfolding** C **by simp**
finally have $r3$: $?C'\langle ts!j \rangle = ?t$ **unfolding** $\langle ts!j = ?D\langle r.\sigma \rangle \rangle$ **by simp**
from $r0 \ r1 \ r2 \ r3$ **show** $?thesis$ **by best**
qed

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** $?f = ?g)$
proof (rule ext)
fix $ab :: 'c * 'b$
obtain $a \ b$ **where** $ab = (a, b)$ **by force**
have $((\lambda(x, y). (x, f \ y)) \circ (\lambda(x, y). (x, g \ y))) \ (a, b) = (\lambda(x, y). (x, (f \circ g) \ y)) \ (a, b)$ **by simp**
then show $?f \ ab = ?g \ ab$ **unfolding** $\langle ab = (a, b) \rangle$ **by simp**
qed

lemma *restrict-singleton*:

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

proof –

have $\sigma \mid s \{x\} = (\lambda y. \text{if } y = x \text{ then } \sigma y \text{ else } \text{Var } y)$ **by** (*simp add: subst-restrict-def*)
then have $\sigma \mid s \{x\} = (\lambda y. \text{if } y = x \text{ then } \sigma x \text{ else } \text{Var } y)$ **by** (*simp cong: if-cong*)
then show *?thesis* **by** (*rule exI[of - σx]*)

qed

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 *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*)

proof

assume *?left*

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

unfolding *rstep-iff-rstep-r-p-s* **by** *blast*

obtain D **where** $D: D = \text{ctxt-of-pos-term } p \ s$ **by** *auto*

with *step* **have** $R\text{rule}: (l, r) \in R$ **and** $s: s = D\langle l \cdot \sigma \rangle$ **and** $t: t = D\langle r \cdot \sigma \rangle$

unfolding *rstep-r-p-s-def* **by** (*force simp: Let-def*)**+**

then show *?right* **unfolding** *rstep-r-c-s-def* **by** *auto*

next

assume *?right*

from this obtain $l \ r \ C \ \sigma$ **where** $(l, r) \in R \wedge (s, t) \in \text{rstep-r-c-s } (l, r) \ C \ \sigma$ **by**

auto

then have *rule*: $(l, r) \in R$ **and** $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$

unfolding *rstep-r-c-s-def* **by** *auto*

show *?left* **unfolding** *rstep-eq-closure* **by** (*auto simp: s t intro: rule*)

qed

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*)

proof

assume *?right*

show *?left*

proof

fix $s \ t$

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

then obtain $l \ r \ C \ \sigma$ **where** $\text{step}: (l, r) \in R \wedge (s, t) \in \text{rstep-r-c-s } (l, r) \ C \ \sigma$

unfolding *rstep-iff-rstep-r-c-s* **by** *best*

then have $R\text{rule}: (l, r) \in R$ **and** $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$

unfolding *rstep-r-c-s-def* **by** (*force*)**+**

from $R\text{rule}$ *<?right>* **obtain** $l' \ r' \ C' \ \sigma'$ **where** $S\text{rule}: (l', r') \in S$ **and** $l: l = C'\langle l' \cdot \sigma' \rangle$ **and** $r: r = C'\langle r' \cdot \sigma' \rangle$

by (*force simp: Let-def*)**+**

let $?D = C \circ_c (C' \cdot_c \sigma)$
 let $?sig = \sigma' \circ_s \sigma$
 have $s2: s = ?D\langle l' \cdot ?sig \rangle$ by (simp add: s l)
 have $t2: t = ?D\langle r' \cdot ?sig \rangle$ by (simp add: t r)
 from $s2\ t2$ have $sStep: (s,t) \in rstep\text{-}r\text{-}c\text{-}s\ (l',r')\ ?D\ ?sig$ unfolding $rstep\text{-}r\text{-}c\text{-}s\text{-}def$
 by force
 with $Srule$ show $(s,t) \in rstep\ S$ by (simp only: $rstep\text{-}iff\text{-}rstep\text{-}r\text{-}c\text{-}s$, blast)
 qed
 next
 assume $?left$
 show $?right$
 proof (rule ccontr)
 assume $\neg ?right$
 from this obtain $l\ r$ where $(l,r) \in R$ and $cond: \forall\ l'\ r'\ C\ \sigma. (l',r') \in S \longrightarrow (l \neq C\langle l' \cdot \sigma \rangle \vee r \neq C\langle r' \cdot \sigma \rangle)$ by blast
 then have $(l,r) \in rstep\ R$ by blast
 with $\langle ?left \rangle$ have $(l,r) \in rstep\ S$ by auto
 with $cond$ show False by (simp only: $rstep\text{-}iff\text{-}rstep\text{-}r\text{-}c\text{-}s$, unfold $rstep\text{-}r\text{-}c\text{-}s\text{-}def$, force)
 qed
 qed

lemma $rstep\text{-}preserves\text{-}funas\text{-}terms\text{-}var\text{-}cond$:

assumes $funas\text{-}trs\ R \subseteq F$ and $funas\text{-}term\ s \subseteq F$ and $(s,t) \in rstep\ R$
 and $wf: \bigwedge\ l\ r. (l,r) \in R \implies vars\text{-}term\ r \subseteq vars\text{-}term\ l$
 shows $funas\text{-}term\ t \subseteq F$
 proof –
 from $\langle (s,t) \in rstep\ R \rangle$ obtain $l\ r\ C\ \sigma$ where $R: (l,r) \in R$
 and $s: s = C\langle l \cdot \sigma \rangle$ and $t: t = C\langle r \cdot \sigma \rangle$ by auto
 from $\langle funas\text{-}trs\ R \subseteq F \rangle$ and R have $funas\text{-}term\ r \subseteq F$
 unfolding $funas\text{-}defs\ [abs\text{-}def]$ by force
 with $wf[OF\ R]$ $\langle funas\text{-}term\ s \subseteq F \rangle$ show $?thesis$ unfolding $s\ t$ by (force simp: $funas\text{-}term\text{-}subst$)
 qed

lemma $rstep\text{-}preserves\text{-}funas\text{-}terms$:

assumes $funas\text{-}trs\ R \subseteq F$ and $funas\text{-}term\ s \subseteq F$ and $(s,t) \in rstep\ R$
 and $wf: wf\text{-}trs\ R$
 shows $funas\text{-}term\ t \subseteq F$
 by (rule $rstep\text{-}preserves\text{-}funas\text{-}terms\text{-}var\text{-}cond[OF\ assms(1-3)]$, insert $wf[unfolded\ wf\text{-}trs\text{-}def]$, auto)

lemma $rsteps\text{-}preserve\text{-}funas\text{-}terms\text{-}var\text{-}cond$:

assumes $F: funas\text{-}trs\ R \subseteq F$ and $s: funas\text{-}term\ s \subseteq F$ and $steps: (s,t) \in (rstep\ R)^*$
 and $wf: \bigwedge\ l\ r. (l,r) \in R \implies vars\text{-}term\ r \subseteq vars\text{-}term\ l$
 shows $funas\text{-}term\ t \subseteq F$
 using $steps$
 proof (induct)

```

    case base then show ?case by (rule s)
next
  case (step t u)
  show ?case by (rule rstep-preserves-funas-terms-var-cond[OF F step(3) step(2)
wf])
qed

lemma rsteps-preserve-funas-terms:
  assumes F: funas-trs R  $\subseteq$  F and s: funas-term s  $\subseteq$  F
  and steps: (s,t)  $\in$  (rstep R)* and wf: wf-trs R
  shows funas-term t  $\subseteq$  F
  by (rule rsteps-preserve-funas-terms-var-cond[OF assms(1-3)], insert wf[unfolded
wf-trs-def], auto)

lemma no-Var-rstep [simp]:
  assumes wf-trs R and (Var x, t)  $\in$  rstep R shows False
  using rstep-imp-Fun[OF assms] by auto

lemma lhs-wf:
  assumes R: (l, r)  $\in$  R and funas-trs R  $\subseteq$  F
  shows funas-term l  $\subseteq$  F
  using assms by (force simp: funas-trs-def funas-rule-def)

lemma rhs-wf:
  assumes R: (l, r)  $\in$  R and funas-trs R  $\subseteq$  F
  shows funas-term r  $\subseteq$  F
  using assms by (force simp: funas-trs-def funas-rule-def)

lemma supt-map-funs-term [intro]:
  assumes t  $\triangleright$  s
  shows map-funs-term fg t  $\triangleright$  map-funs-term fg s
using assms
proof (induct)
  case (arg s ss f)
  then have map-funs-term fg s  $\in$  set(map (map-funs-term fg) ss) by simp
  then show ?case unfolding term.map by (rule supt.arg)
next
  case (subt s ss u f)
  then have map-funs-term fg s  $\in$  set(map (map-funs-term fg) ss) by simp
  with  $\langle$ map-funs-term fg s  $\triangleright$  map-funs-term fg u $\rangle$  show ?case
  unfolding term.map by (metis supt.subt)
qed

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

```

proof –
 from *assms* obtain $l\ r\ p\ \sigma$
 where $rstep\text{-}r\text{-}p\text{-}s$: $(Fun\ f\ ss,\ t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ (l,\ r)\ p\ \sigma$
 unfolding $rstep\text{-}iff\text{-}rstep\text{-}r\text{-}p\text{-}s$ by *auto*
 let $?C = ctxt\text{-}of\text{-}pos\text{-}term\ p\ (Fun\ f\ ss)$
 from $rstep\text{-}r\text{-}p\text{-}s$ have $p \in poss\ (Fun\ f\ ss)$ and $(l,\ r) \in R$
 and $?C\langle l \cdot \sigma \rangle = Fun\ f\ ss$ and $?C\langle r \cdot \sigma \rangle = t$ unfolding $rstep\text{-}r\text{-}p\text{-}s\text{-}def\ Let\text{-}def$
 by *auto*
 have $\exists i\ q.\ p = i\#q$
proof (*cases* p)
 case *Cons* then show *?thesis* by *auto*
 next
 case *Nil*
 have $?C = \square$ unfolding *Nil* by *simp*
 with $\langle ?C\langle l \cdot \sigma \rangle = Fun\ f\ ss \rangle$ have $l \cdot \sigma = Fun\ f\ ss$ by *simp*
 have $\forall (l,r) \in R.\ \exists f\ ss.\ l = Fun\ f\ ss$
proof (*intro ballI impI*)
 fix lr assume $lr \in R$
 with *wf* have $\forall x.\ fst\ lr \neq Var\ x$ by *auto*
 then have $\exists f\ ss.\ (fst\ lr) = Fun\ f\ ss$ by (*cases* $fst\ lr$) *auto*
 then show $(\lambda(l,r).\ \exists f\ ss.\ l = Fun\ f\ ss)\ lr$ by *auto*
 qed
 with $\langle (l,r) \in R \rangle$ obtain $g\ ts$ where $l = Fun\ g\ ts$ unfolding *wf\text{-}trs\text{-}def* by *best*
 with $\langle l \cdot \sigma = Fun\ f\ ss \rangle\ \langle l = Fun\ g\ ts \rangle$ and $\langle (l,\ r) \in R \rangle\ ndef$
 show *?thesis* unfolding *defined\text{-}def* by *auto*
 qed
 then obtain $i\ q$ where $p = i\#q$ by *auto*
 from $\langle p \in poss(Fun\ f\ ss) \rangle$ have $i < length\ ss$ and $q \in poss(ss!i)$ unfolding $\langle p = i\#q \rangle$ by *auto*
 let $?D = ctxt\text{-}of\text{-}pos\text{-}term\ q\ (ss!i)$
 have $C: ?C = More\ f\ (take\ i\ ss)\ ?D\ (drop\ (Suc\ i)\ ss)$ unfolding $\langle p = i\#q \rangle$ by *auto*
 from $\langle ?C\langle l \cdot \sigma \rangle = Fun\ f\ ss \rangle$ have $take\ i\ ss @ ?D\langle l \cdot \sigma \rangle \# drop\ (Suc\ i)\ ss = ss$ unfolding *C* by *auto*
 then have $(take\ i\ ss @ ?D\langle l \cdot \sigma \rangle \# drop\ (Suc\ i)\ ss)!i = ss!i$ by *simp*
 with $\langle i < length\ ss \rangle$ have $?D\langle l \cdot \sigma \rangle = ss!i$ using *nth\text{-}append\text{-}take* [where $xs = ss$ and $i = i$] by *auto*
 have $t: t = Fun\ f\ (take\ i\ ss @ ?D\langle r \cdot \sigma \rangle \# drop\ (Suc\ i)\ ss)$ unfolding $\langle ?C\langle r \cdot \sigma \rangle = t \rangle$ [symmetric] *C* by *simp*
 from $\langle i < length\ ss \rangle$ have $t|-[i] = ?D\langle r \cdot \sigma \rangle$ unfolding *t* unfolding *subt\text{-}at\text{-}sims* using *nth\text{-}append\text{-}take* [where $xs = ss$ and $i = i$] by *auto*
 from $\langle q \in poss(ss!i) \rangle$ and $\langle (l,r) \in R \rangle$
 and $\langle ?D\langle l \cdot \sigma \rangle = ss!i \rangle$ and $\langle t|-[i] = ?D\langle r \cdot \sigma \rangle \rangle$ [symmetric]
 have $(ss!i, t|-[i]) \in rstep\text{-}r\text{-}p\text{-}s\ R\ (l,r)\ q\ \sigma$ unfolding $rstep\text{-}r\text{-}p\text{-}s\text{-}def\ Let\text{-}def$ by *auto*
 then have $(ss!i, t|-[i]) \in rstep\ R$ unfolding $rstep\text{-}iff\text{-}rstep\text{-}r\text{-}p\text{-}s$ by *auto*
 from $\langle i < length\ ss \rangle$ and *this* and *t* show *?thesis* unfolding $\langle t|-[i] = ?D\langle r \cdot \sigma \rangle \rangle$ [symmetric] by *auto*
 qed

lemma *nondef-root-imp-arg-steps*:
assumes $(Fun\ f\ ss, t) \in (rstep\ R)^*$
and $wf: \forall (l, r) \in R. is-Fun\ l$
and $\neg defined\ R\ (f, length\ ss)$
shows $\exists ts. length\ ts = length\ ss \wedge t = Fun\ f\ ts \wedge (\forall i < length\ ss. (ss!i, ts!i) \in (rstep\ R)^*)$
proof –
from *assms* **obtain** n **where** $(Fun\ f\ ss, t) \in (rstep\ R)^{\sim n}$
using *rtrancl-imp-relpow* **by** *best*
then show *?thesis*
proof (*induct n arbitrary: t*)
case 0 **then show** *?case* **by** *auto*
next
case (*Suc n*)
then obtain u **where** $(Fun\ f\ ss, u) \in (rstep\ R)^{\sim n}$ **and** $(u, t) \in rstep\ R$ **by** *auto*
with *Suc* **obtain** ts **where** $IH1: length\ ts = length\ ss$ **and** $IH2: u = Fun\ f\ ts$
and $IH3: \forall i < length\ ss. (ss!i, ts!i) \in (rstep\ R)^*$ **by** *auto*
from $\langle (u, t) \in rstep\ R \rangle$ **have** $(Fun\ f\ ts, t) \in rstep\ R$ **unfolding** $\langle u = Fun\ f\ ts \rangle$.
from *nondef-root-imp-arg-step* [*OF this wf* $\langle \neg defined\ R\ (f, length\ ss) \rangle$] [*simplified* $IH1$ [*symmetric*]]
obtain j **where** $j < length\ ts$
and $(ts!j, t|-[j]) \in rstep\ R$
and $B: t = Fun\ f\ (take\ j\ ts @ (t|-[j]) \# drop\ (Suc\ j)\ ts)$ (**is** $t = Fun\ f\ ?ts$) **by** *auto*
from $\langle j < length\ ts \rangle$ **have** $length\ ?ts = length\ ts$ **by** *auto*
then have $A: length\ ?ts = length\ ss$ **unfolding** $\langle length\ ts = length\ ss \rangle$.
have $C: \forall i < length\ ss. (ss!i, ?ts!i) \in (rstep\ R)^*$
proof (*intro allI, intro impI*)
fix i **assume** $i < length\ ss$
from $\langle i < length\ ss \rangle$ **and** $IH3$ **have** $(ss!i, ts!i) \in (rstep\ R)^*$ **by** *auto*
have $i = j \vee i \neq j$ **by** *auto*
then show $(ss!i, ?ts!i) \in (rstep\ R)^*$
proof
assume $i = j$
from $\langle j < length\ ts \rangle$ **have** $j \leq length\ ts$ **by** *simp*
from *nth-append-take* [*OF this*] **have** $?ts!j = t|-[j]$ **by** *simp*
from $\langle (ts!j, t|-[j]) \in rstep\ R \rangle$ **have** $(ts!i, t|-[i]) \in rstep\ R$ **unfolding** $\langle i = j \rangle$.
with $\langle (ss!i, ts!i) \in (rstep\ R)^* \rangle$ **show** *?thesis* **unfolding** $\langle i = j \rangle$ **unfolding** $\langle ?ts!j = t|-[j] \rangle$ **by** *auto*
next
assume $i \neq j$
from $\langle i < length\ ss \rangle$ **have** $i \leq length\ ts$ **unfolding** $\langle length\ ts = length\ ss \rangle$
by *simp*
from $\langle j < length\ ts \rangle$ **have** $j \leq length\ ts$ **by** *simp*
from *nth-append-take-drop-is-nth-conv* [*OF* $\langle i \leq length\ ts \rangle$ $\langle j \leq length\ ts \rangle$ $\langle i \neq j \rangle$]
have $?ts!i = ts!i$ **by** *simp*

with $\langle (ss!i, ts!i) \in (rstep\ R)^* \rangle$ **show** *?thesis* **by** *auto*
 qed
 qed
 from *A* and *B* and *C* **show** *?case* **by** *blast*
 qed
 qed
 lemma *rstep-imp-nrrstep*:
 assumes *is-Fun s* and $\neg\ defined\ R\ (the\ (root\ s))$ and $\forall (l,r) \in R. is-Fun\ l$
 and $(s, t) \in rstep\ R$
 shows $(s, t) \in nrrstep\ R$
 proof -
 from $\langle is-Fun\ s \rangle$ **obtain** *f ss* **where** $s = Fun\ f\ ss$ **by** *(cases s) auto*
 with *assms* **have** *undef*: $\neg\ defined\ R\ (f, length\ ss)$ **by** *simp*
 from *assms* **have** *non-var*: $\forall (l, r) \in R. is-Fun\ l$ **by** *auto*
 from *nondef-root-imp-arg-step*[*OF* $\langle (s, t) \in rstep\ R \rangle$ [*unfolded s*] *non-var undef*]
obtain *i* **where** $i < length\ ss$ and *step*: $(ss!\ i, t\ |- [i]) \in rstep\ R$
 and *t*: $t = Fun\ f\ (take\ i\ ss\ @\ (t\ |- [i])\ \# drop\ (Suc\ i)\ ss)$ **by** *auto*
 from *step* **obtain** *C l r σ* **where** $(l, r) \in R$ and *lhs*: $ss!\ i = C\langle l \cdot \sigma \rangle$
 and *rhs*: $t\ |- [i] = C\langle r \cdot \sigma \rangle$ **by** *auto*
 let *?C* = *More f (take i ss) C (drop (Suc i) ss)*
 have $(l, r) \in R$ **by** *fact*
 moreover **have** *?C* $\neq \square$ **by** *simp*
 moreover **have** $s = ?C\langle l \cdot \sigma \rangle$
 proof -
 have $s = Fun\ f\ (take\ i\ ss\ @\ ss!\ i\ \# drop\ (Suc\ i)\ ss)$
 using *id-take-nth-drop*[*OF* $\langle i < length\ ss \rangle$] **unfolding** *s* **by** *simp*
 also **have** $\dots = ?C\langle l \cdot \sigma \rangle$ **by** *(simp add: lhs)*
 finally **show** *?thesis* .
 qed
 moreover **have** $t = ?C\langle r \cdot \sigma \rangle$
 proof -
 have $t = Fun\ f\ (take\ i\ ss\ @\ t\ |- [i]\ \# drop\ (Suc\ i)\ ss)$ **by** *fact*
 also **have** $\dots = Fun\ f\ (take\ i\ ss\ @\ C\langle r \cdot \sigma \rangle\ \# drop\ (Suc\ i)\ ss)$ **by** *(simp add: rhs)*
 finally **show** *?thesis* **by** *simp*
 qed
 ultimately **show** $(s, t) \in nrrstep\ R$ **unfolding** *nrrstep-def'* **by** *blast*
 qed
 lemma *rsteps-imp-nrrsteps*:
 assumes *is-Fun s* and $\neg\ defined\ R\ (the\ (root\ s))$
 and *no-vars*: $\forall (l, r) \in R. is-Fun\ l$
 and $(s, t) \in (rstep\ R)^*$
 shows $(s, t) \in (nrrstep\ R)^*$
 using $\langle (s, t) \in (rstep\ R)^* \rangle$
 proof (*induct*)
 case *base* **show** *?case* **by** *simp*
 next

```

case (step u v)
from assms obtain f ss where s: s = Fun f ss by (induct s) auto
from nrrsteps-preserve-root[OF  $\langle (s, u) \in (\text{nrrstep } R)^* \rangle$  [unfolded s]]
  obtain ts where u: u = Fun f ts by auto
from nrrsteps-equiv-num-args[OF  $\langle (s, u) \in (\text{nrrstep } R)^* \rangle$  [unfolded s]]
  have len: length ss = length ts unfolding u by simp
have is-Fun u by (simp add: u)
have undef:  $\neg \text{defined } R \text{ (the (root u))}$ 
  using  $\langle \neg \text{defined } R \text{ (the (root s))} \rangle$ 
  unfolding s u by (simp add: len)
have (u, v)  $\in \text{nrrstep } R$ 
  using rstep-imp-nrrstep[OF  $\langle \text{is-Fun } u \rangle \text{ undef no-vars}$ ] step
  by simp
with step show ?case by auto
qed

```

```

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  $\neg (\text{SN-on } (\text{rstep } R) \{t\})$ 
proof (rule steps-imp-not-SN-on)
  fix t :: ('f, 'v) term
  let ?yt = subst y t
  show (t, r  $\cdot$  ?yt)  $\in \text{rstep } R$ 
    by (rule rstepI[OF assms, where C =  $\square$  and  $\sigma = ?yt$ ], auto simp: subst-def)
qed

```

```

lemma not-SN-subt-imp-not-SN:
  assumes ctxt: ctxt.closed R and SN:  $\neg \text{SN-on } R \{t\}$  and sub: s  $\supseteq$  t
  shows  $\neg \text{SN-on } R \{s\}$ 
  using ctxt-closed-SN-on-subt[OF ctxt - sub] SN
  by auto

```

```

lemma root-Some:
  assumes root t = Some fn
  obtains ss where length ss = snd fn and t = Fun (fst fn) ss
using assms by (induct t) auto

```

```

lemma map-funs-rule-power:
  fixes f :: 'f  $\Rightarrow$  'f
  shows ((map-funs-rule f)  $\frown$  n) = map-funs-rule (f  $\frown$  n)
proof (rule sym, intro ext, clarify)
  fix l r :: ('f, 'v) term
  show map-funs-rule (f  $\frown$  n) (l, r) = (map-funs-rule f  $\frown$  n) (l, r)
proof (induct n)
  case 0
  show ?case by (simp add: term.map-ident)
next
  case (Suc n)

```

```

  have map-funs-rule (f  $\sim$  Suc n) (l,r) = map-funs-rule f (map-funs-rule (f  $\sim$ 
n) (l,r))
    by (simp add: map-funs-term-comp)
  also have ... = map-funs-rule f ((map-funs-rule f  $\sim$  n) (l,r)) unfolding Suc
..
  also have ... = (map-funs-rule f  $\sim$  (Suc n)) (l,r) by simp
  finally show ?case .
qed
qed

```

lemma *map-funs-trs-power*:

```

  fixes f :: 'f  $\Rightarrow$  'f
  shows map-funs-trs f  $\sim$  n = map-funs-trs (f  $\sim$  n)
proof
  fix R :: ('f, 'v) trs
  have map-funs-rule (f  $\sim$  n) ' R = (map-funs-rule f  $\sim$  n) ' R unfolding
map-funs-rule-power ..
  also have ... = (( $\lambda$  R. map-funs-trs f R)  $\sim$  n) R unfolding map-funs-trs.simps
  apply (induct n)
  apply simp
  by (metis comp-apply funpow.simps(2) image-comp)
  finally have map-funs-rule (f  $\sim$  n) ' R = (map-funs-trs f  $\sim$  n) R .
  then show (map-funs-trs f  $\sim$  n) R = map-funs-trs (f  $\sim$  n) R
    by (simp add: map-funs-trs.simps)
qed

```

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  $\exists t \trianglelefteq s. t \in Tinf\ R$ 
proof -
  let ?S = {t | t. s  $\triangleright$  t  $\wedge$   $\neg SN\text{-on}\ R\ \{t\}$ }
  from assms have s: s  $\in$  ?S by auto
  from mp[OF spec[OF spec[OF SN-imp-minimal[OF SN-supt]]] s]
  obtain t where st: s  $\triangleright$  t and nSN:  $\neg SN\text{-on}\ R\ \{t\}$ 
    and min:  $\forall u. (t,u) \in supt \longrightarrow u \notin ?S$  by auto
  have t  $\in Tinf\ R$  unfolding Tinf-def
proof (intro CollectI allI impI conjI nSN)
  fix u
  assume u: t  $\triangleright$  u
  from u st have s  $\triangleright$  u using supteq-supt-trans by auto
  with min u show SN-on R {u} by auto
qed
with st show ?thesis by auto
qed

```

lemma *not-SN-imp-Tinf*:
assumes $\neg SN\ R$ **shows** $\exists t. t \in Tinf\ R$
using *assms not-SN-imp-subt-Tinf unfolding SN-on-def by blast*

lemma *ctxt-of-pos-term-map-funs-term-conv* [iff]:
assumes $p \in poss\ s$
shows $map-funs-ctxt\ fg\ (ctxt-of-pos-term\ p\ s) = (ctxt-of-pos-term\ p\ (map-funs-term\ fg\ s))$
using *assms*
proof (*induct s arbitrary: p*)
case (*Var x*) **then show** ?case **by simp**
next
case (*Fun f ss*) **then show** ?case
proof (*cases p*)
case Nil **then show** ?thesis **by simp**
next
case (*Cons i q*)
with $\langle p \in poss(Fun\ f\ ss) \rangle$ **have** $i < length\ ss$ **and** $q \in poss(ss!i)$ **unfolding** *Cons*
poss.simps **by auto**
then have $ss!i \in set\ ss$ **by simp**
with *Fun* **and** $\langle q \in poss(ss!i) \rangle$
have *IH*: $map-funs-ctxt\ fg\ (ctxt-of-pos-term\ q\ (ss!i)) = ctxt-of-pos-term\ q\ (map-funs-term\ fg\ (ss!i))$ **by simp**
have $map-funs-ctxt\ fg\ (ctxt-of-pos-term\ p\ (Fun\ f\ ss)) = map-funs-ctxt\ fg\ (ctxt-of-pos-term\ (i\#q)\ (Fun\ f\ ss))$ **unfolding** *Cons* **by simp**
also have $\dots = map-funs-ctxt\ fg\ (More\ f\ (take\ i\ ss)\ (ctxt-of-pos-term\ q\ (ss!i))\ (drop\ (Suc\ i)\ ss))$ **by simp**
also have $\dots = More\ (fg\ f)\ (map\ (map-funs-term\ fg)\ (take\ i\ ss))\ (map-funs-ctxt\ fg\ (ctxt-of-pos-term\ q\ (ss!i)))\ (map\ (map-funs-term\ fg)\ (drop\ (Suc\ i)\ ss))$ **by simp**
also have $\dots = More\ (fg\ f)\ (map\ (map-funs-term\ fg)\ (take\ i\ ss))\ (ctxt-of-pos-term\ q\ (map-funs-term\ fg\ (ss!i)))\ (map\ (map-funs-term\ fg)\ (drop\ (Suc\ i)\ ss))$ **unfolding** *IH* **by simp**
also have $\dots = More\ (fg\ f)\ (take\ i\ (map\ (map-funs-term\ fg)\ ss))\ (ctxt-of-pos-term\ q\ (map\ (map-funs-term\ fg)\ ss!i))\ (drop\ (Suc\ i)\ (map\ (map-funs-term\ fg)\ ss))$ **unfolding** *nth-map[OF <i < length ss>,symmetric]* *take-map drop-map nth-map* **by simp**
finally show ?thesis **unfolding** *Cons* **by simp**
qed
qed

lemma *var-rewrite-imp-not-SN*:
assumes *sn*: $SN-on\ (rstep\ R)\ \{u\}$ **and** *step*: $(t, s) \in rstep\ R$
shows *is-Fun* *t*
using *assms*
proof (*cases t*)
case (*Fun f ts*) **then show** ?thesis **by simp**
next
case (*Var x*)
from *step* **obtain** $l\ r\ p\ \sigma$ **where** $(Var\ x, s) \in rstep-r-p-s\ R\ (l, r)\ p\ \sigma$ **unfolding**

Var *rstep-iff-rstep-r-p-s* **by** *best*
then have $l \cdot \sigma = \text{Var } x$ **and** *rule*: $(l, r) \in R$ **unfolding** *rstep-r-p-s-def* **by** (*auto simp: Let-def*)
from this obtain *y* **where** $l = \text{Var } y$ (**is** $- = ?y$) **by** (*cases l, auto*)
with rule have $(?y, r) \in R$ **by** *auto*
then have $\neg (SN\text{-on } (rstep\ R) \ \{u\})$ **by** (*rule left-var-imp-not-SN*)
with sn show *?thesis* **by** *blast*
qed

lemma *rstep-id*: $rstep\ Id = Id$ **by** *auto*

lemma *map-funs-rule-id* [*simp*]: $map\ funs\ rule\ id = id$
by (*intro ext, auto*)

lemma *map-funs-trs-id* [*simp*]: $map\ funs\ trs\ id = id$
by (*intro ext, auto simp: map-funs-trs.simps*)

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*: $sig\ step\ F\ (R \cup S) = sig\ step\ F\ R \cup sig\ step\ F\ S$
unfolding *sig-step-def* **by** *auto*

lemma *sig-step-UNIV*: $sig\ step\ UNIV\ R = R$ **unfolding** *sig-step-def* **by** *simp*

lemma *sig-stepI*[*intro*]: $(a, b) \in R \implies funas\ term\ a \subseteq F \implies funas\ term\ b \subseteq F$
 $\implies (a, b) \in sig\ step\ F\ R$ **unfolding** *sig-step-def* **by** *auto*

lemma *sig-stepE*[*elim, consumes 1*]: $(a, b) \in sig\ step\ F\ R \implies [(a, b) \in R \implies funas\ term\ a \subseteq F \implies funas\ term\ b \subseteq F \implies P] \implies P$ **unfolding** *sig-step-def* **by** *auto*

lemma *all-ctxt-closed-sig-rsteps* [*intro*]:
fixes $R :: ('f, 'v)\ trs$
shows $all\ ctxt\ closed\ F\ ((sig\ step\ F\ (rstep\ R))^*)$ (**is** $all\ ctxt\ closed - (?R^*)$)
proof (*rule trans-ctxt-sig-imp-all-ctxt-closed*)
fix $C :: ('f, 'v)\ ctxt$ **and** $s\ t :: ('f, 'v)\ term$
assume $C: funas\ ctxt\ C \subseteq F$
and $s: funas\ term\ s \subseteq F$
and $t: funas\ term\ t \subseteq F$
and $steps: (s, t) \in ?R^*$
from *steps*
show $(C \langle s \rangle, C \langle t \rangle) \in ?R^*$
proof (*induct*)
case (*step t u*)
from *step(2)* **have** $tu: (t, u) \in rstep\ R$ **and** $t: funas\ term\ t \subseteq F$ **and** $u: funas\ term\ u \subseteq F$ **by** *auto*
have $(C \langle t \rangle, C \langle u \rangle) \in ?R$ **by** (*rule sig-stepI[OF rstep-ctxt[OF tu]], insert C t u, auto*)

```

    with step(3) show ?case by auto
  qed auto
qed (auto intro: trans-rtranc1)

lemma wf-loop-imp-sig-ctxt-rel-not-SN:
  assumes R:  $(l, C\langle l \rangle) \in R$  and wf-l:  $\text{funas-term } l \subseteq F$ 
  and wf-C:  $\text{funas-ctxt } C \subseteq F$ 
  and ctxt:  $\text{ctxt.closed } R$ 
  shows  $\neg \text{SN-on } (\text{sig-step } F \ R) \ \{l\}$ 
proof -
  let ?t =  $\lambda i. (C\hat{\sim}i)\langle l \rangle$ 
  have  $\forall i. \text{funas-term } (?t \ i) \subseteq F$ 
  proof
    fix i show  $\text{funas-term } (?t \ i) \subseteq F$  unfolding funas-term-ctxt-apply
      by (rule Un-least[OF wf-l], induct i, insert wf-C, auto)
  qed
  moreover have  $\forall i. (?t \ i, ?t(Suc \ i)) \in R$ 
  proof
    fix i
    show  $(?t \ i, ?t(Suc \ i)) \in R$ 
    proof (induct i)
      case 0 with R show ?case by auto
    next
      case (Suc i)
      from ctxt.closedD[OF ctxt Suc, of C]
      show ?case by simp
    qed
  qed
  ultimately have steps:  $\forall i. (?t \ i, ?t(Suc \ i)) \in \text{sig-step } F \ R$  unfolding sig-step-def
  by blast
  show ?thesis unfolding SN-defs
    by (simp, intro exI[of - ?t], simp only: steps, simp)
qed

lemma lhs-var-imp-sig-step-not-SN-on:
  assumes x:  $(Var \ x, r) \in R$  and F:  $\text{funas-trs } R \subseteq F$ 
  shows  $\neg \text{SN-on } (\text{sig-step } F \ (\text{rstep } R)) \ \{Var \ x\}$ 
proof -
  let ? $\sigma$  =  $(\lambda x. r)$ 
  let ?t =  $\lambda i. (?\sigma \ \smile i) \ x$ 
  obtain t where t:  $t = ?t$  by auto
  from rhs-wf[OF x F] have wf-r:  $\text{funas-term } r \subseteq F$  .
  {
    fix i
    have  $\text{funas-term } (?t \ i) \subseteq F$ 
    proof (induct i)
      case 0 show ?case using wf-r by auto
    next
      case (Suc i)

```

have $?t (Suc\ i) = ?t\ i \cdot ?\sigma$ **unfolding** *subst-power-Suc subst-compose-def* **by**
simp
also have $funas-term\ \dots \subseteq F$ **unfolding** *funas-term-subst[of ?t i]*
using *Suc wf-r* **by** *auto*
finally show $?case$.
qed
} **note** $wf-t = this$
{
fix i
have $(t\ i, t\ (Suc\ i)) \in (sig-step\ F\ (rstep\ R))$ **unfolding** t
by $(rule\ sig-stepI[OF\ rstepI[OF\ x, of\ -\ \square\ ?\sigma\ \sim i]\ wf-t\ wf-t], auto\ simp:$
subst-compose-def)
} **note** $steps = this$
have $x: t\ 0 = Var\ x$ **unfolding** t **by** *simp*
with $steps$ **show** $?thesis$ **unfolding** *SN-defs not-not*
by $(intro\ exI[of\ -\ t], auto)$
qed

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 $\neg SN-on\ (sig-step\ F\ (rstep\ R))\ \{l\}$
proof –
from $free$ **obtain** x **where** $x: x \in vars-term\ r - vars-term\ l$ **by** *auto*
then have $x \in vars-term\ r$ **by** *simp*
from *supteq-Var[OF this]* **have** $r \supseteq Var\ x$.
then obtain C **where** $r: C\langle Var\ x \rangle = r$ **by** *auto*
let $?\sigma = \lambda y. if\ y = x\ then\ l\ else\ Var\ y$
let $?t = \lambda i. ((C \cdot_c ?\sigma) \hat{\sim} i)\langle l \rangle$
from *rhs-wf[OF R]* F **have** $wf-r: funas-term\ r \subseteq F$ **by** *fast*
from *lhs-wf[OF R]* F **have** $wf-l: funas-term\ l \subseteq F$ **by** *fast*
from *wf-r[unfolded r[symmetric]]*
have $wf-C: funas-ctxt\ C \subseteq F$ **by** *simp*
from x **have** $neg: \forall y \in vars-term\ l. y \neq x$ **by** *auto*
have $l \cdot ?\sigma = l \cdot Var$
by $(rule\ term-subst-eq, insert\ neg, auto)$
then have $l: l \cdot ?\sigma = l$ **by** *simp*
have $wf-C: funas-ctxt\ (C \cdot_c ?\sigma) \subseteq F$ **using** *wf-C wf-l*
by *simp*
have $rsigma: r \cdot ?\sigma = (C \cdot_c ?\sigma)\langle l \rangle$ **unfolding** *r[symmetric]* **by** *simp*
from R **have** $lr: (l \cdot ?\sigma, r \cdot ?\sigma) \in rstep\ R$ **by** *auto*
then have $lr: (l, (C \cdot_c ?\sigma)\langle l \rangle) \in rstep\ R$ **unfolding** l **unfolding** $rsigma$.
show $?thesis$
by $(rule\ wf-loop-imp-sig-ctxt-rel-not-SN[OF\ lr\ wf-l\ wf-C\ ctxt-closed-rstep])$
qed

lemma *lhs-var-imp-rstep-not-SN*: **assumes** $(Var\ x, r) \in R$ **shows** $\neg SN(rstep\ R)$
using *lhs-var-imp-sig-step-not-SN-on[OF assms subset-refl]* **unfolding** *sig-step-def*
SN-defs **by** *blast*

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 $\neg \text{SN-on } (\text{rstep } R) \{l\}$
using *rhs-free-vars-imp-sig-step-not-SN*[*OF assms subset-refl*] **unfolding** *sig-step-def*
SN-defs **by** *blast*

lemma *free-right-rewrite-imp-not-SN*:
assumes *step*: $(t,s) \in \text{rstep-r-p-s } R (l,r) \text{ } p \text{ } \sigma$
and *vars*: $\neg \text{vars-term } l \supseteq \text{vars-term } r$
shows $\neg \text{SN-on } (\text{rstep } R) \{t\}$
proof
assume *SN*: $\text{SN-on } (\text{rstep } R) \{t\}$
let $?C = \text{ctxt-of-pos-term } p \text{ } t$
from *step* **have** *left*: $?C \langle l \cdot \sigma \rangle = t$ (**is** $?t = t$) **and** *right*: $?C \langle r \cdot \sigma \rangle = s$ **and**
pos: $p \in \text{pos } t$
and *rule*: $(l,r) \in R$
unfolding *rstep-r-p-s-def* **by** (*auto simp: Let-def*)
from *rhs-free-vars-imp-rstep-not-SN*[*OF rule vars*] **have** $n\text{SN} : \neg \text{SN-on } (\text{rstep } R)$
 $\{l\}$ **by** *simp*
from *SN-imp-SN-subt*[*OF SN ctxt-imp-supteq*[*of ?C l \cdot \sigma, simplified left*]]
have *SN*: $\text{SN-on } (\text{rstep } R) \{l \cdot \sigma\}$.
from *SNinstance-imp-SN*[*OF SN*] $n\text{SN}$ **show** *False* **by** *simp*
qed

lemma *not-SN-on-rstep-subst-apply-term*[*intro*]:
assumes $\neg \text{SN-on } (\text{rstep } R) \{t\}$ **shows** $\neg \text{SN-on } (\text{rstep } R) \{t \cdot \sigma\}$
using *assms* **unfolding** *SN-on-def* **by** *best*

lemma *SN-rstep-imp-wf-trs*: **assumes** *SN* $(\text{rstep } R)$ **shows** *wf-trs* *R*
proof (*rule ccontr*)
assume $\neg \text{wf-trs } R$
then obtain $l \text{ } r$ **where** *R*: $(l,r) \in R$
and *not-wf*: $(\forall f \text{ } ts. l \neq \text{Fun } f \text{ } ts) \vee \neg(\text{vars-term } r \subseteq \text{vars-term } l)$ **unfolding**
wf-trs-def
by *auto*
from *not-wf* **have** $\neg \text{SN } (\text{rstep } R)$
proof
assume *free*: $\neg \text{vars-term } r \subseteq \text{vars-term } l$
from *rhs-free-vars-imp-rstep-not-SN*[*OF R free*] **show** *?thesis* **unfolding** *SN-defs*
by *auto*
next
assume $\forall f \text{ } ts. l \neq \text{Fun } f \text{ } ts$
then obtain x **where** $l : l = \text{Var } x$ **by** (*cases l*) *auto*
with *R* **have** $(\text{Var } x, r) \in R$ **unfolding** *l* **by** *simp*
from *lhs-var-imp-rstep-not-SN*[*OF this*] **show** *?thesis* **by** *simp*
qed
with *assms* **show** *False* **by** *blast*
qed

lemma *SN-sig-step-imp-wf-trs*: **assumes** *SN*: *SN (sig-step F (rstep R))* **and** *F*: *funas-trs R ⊆ F* **shows** *wf-trs R*

proof (*rule ccontr*)

assume $\neg \text{wf-trs } R$

then obtain *l r* **where** *R*: $(l, r) \in R$

and *not-wf*: $(\forall f \text{ ts. } l \neq \text{Fun } f \text{ ts}) \vee \neg(\text{vars-term } r \subseteq \text{vars-term } l)$ **unfolding** *wf-trs-def*

by *auto*

from *not-wf* **have** $\neg \text{SN (sig-step } F \text{ (rstep } R))$

proof

assume *free*: $\neg \text{vars-term } r \subseteq \text{vars-term } l$

from *rhs-free-vars-imp-sig-step-not-SN*[*OF R free F*] **show** *?thesis* **unfolding** *SN-on-def* **by** *auto*

next

assume $\forall f \text{ ts. } l \neq \text{Fun } f \text{ ts}$

then obtain *x* **where** *l*: $l = \text{Var } x$ **by** (*cases l*) *auto*

with *R* **have** $(\text{Var } x, r) \in R$ **unfolding** *l* **by** *simp*

from *lhs-var-imp-sig-step-not-SN-on*[*OF this F*] **show** *?thesis*

unfolding *SN-on-def* **by** *auto*

qed

with *assms* **show** *False* **by** *blast*

qed

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

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

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

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

shows *P*

proof –

from *rstep-imp-C-s-r*[*OF rstep*] **obtain** *C* *σ* *l r*

where *R*: $(l, r) \in R$ **and** *s*: $C \langle l \cdot \sigma \rangle = s$ **and** *t*: $C \langle r \cdot \sigma \rangle = t$ **by** *fast*

show *?thesis* **proof** (*cases C*)

case *Hole*

from *s t* **have** $l \cdot \sigma = s$ **and** $r \cdot \sigma = t$ **by** (*auto simp: Hole*)

with *R* **show** *?thesis* **by** (*rule root*)

next

case (*More f ss1 D ss2*)

let *?u* = $D \langle l \cdot \sigma \rangle$

let *?v* = $D \langle r \cdot \sigma \rangle$

have $s = \text{Fun } f (ss1 @ ?u \# ss2)$ **by** (*simp add: More s[symmetric]*)

moreover $t = \text{Fun } f (ss1 @ ?v \# ss2)$ **by** (*simp add: More t[symmetric]*)

moreover $(?u, ?v) \in \text{rstep } R$ **using** *R* **by** *auto*

ultimately **show** *?thesis* **by** (*rule nonroot*)

qed

qed

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

proof

assume $(\text{Var } x, t) \in \text{rstep } R$
from $\text{rstep-imp-C-s-r}[OF \text{ this}]$ **obtain** $C \ l \ r \ \sigma$
where $R: (l, r) \in R$ **and** $\text{lhs}: \text{Var } x = C \langle l \cdot \sigma \rangle$ **by** *fast*
from lhs **have** $\text{Var } x = l \cdot \sigma$ **by** $(\text{induct } C)$ *auto*
then obtain y **where** $l: l = \text{Var } y$ **by** $(\text{induct } l)$ *auto*
from $\text{wf } R$ **obtain** $f \ ss$ **where** $l = \text{Fun } f \ ss$ **unfolding** wf-trs-def **by** *best*
with l **show** *False* **by** *simp*
qed

lemma $\text{rstep-cases-Fun}'[\text{consumes } 2, \text{case-names root nonroot}]$:

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

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

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

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

shows P

using rstep **proof** $(\text{cases rule: rstep-cases}')$

case $(\text{root } l \ r \ \sigma)$

with wf **obtain** $g \ ls$ **where** $l: l = \text{Fun } g \ ls$ **unfolding** wf-trs-def **by** *best*

from root **have** $[simp]: g = f$ **unfolding** l **by** *simp*

from root **have** $(\text{Fun } f \ ls, r) \in R$ **and** $\text{map } (\lambda t. t \cdot \sigma) \ ls = ss$ **and** $r \cdot \sigma = t$ **unfolding** l **by** *auto*

then show $?thesis$ **by** $(\text{rule } \text{root}')$

next

case $(\text{nonroot } g \ ss1 \ u \ ss2 \ v)$

then have $[simp]: g = f$ **and** $\text{args}: ss = ss1 \ @ \ u \ \# \ ss2$ **by** *auto*

let $?i = \text{length } ss1$

from args **have** $ss1: \text{take } ?i \ ss = ss1$ **by** *simp*

from args **have** $\text{drop } ?i \ ss = u \ \# \ ss2$ **by** *simp*

then have $\text{drop } (\text{Suc } 0) (\text{drop } ?i \ ss) = ss2$ **by** *simp*

then have $ss2: \text{drop } (\text{Suc } ?i) \ ss = ss2$ **by** *simp*

from args **have** $\text{len}: ?i < \text{length } ss$ **by** *simp*

from $\text{id-take-nth-drop}[OF \ \text{len}]$ **have** $ss = \text{take } ?i \ ss \ @ \ ss! ?i \ \# \ \text{drop } (\text{Suc } ?i) \ ss$ **by** *simp*

then have $u: ss! ?i = u$ **unfolding** args **unfolding** $ss1[\text{unfolded args}] \ ss2[\text{unfolded args}]$ **by** *simp*

from nonroot **have** $t = \text{Fun } f \ (\text{take } ?i \ ss @ v \# \text{drop } (\text{Suc } ?i) \ ss)$ **unfolding** $ss1 \ ss2$ **by** *simp*

moreover from nonroot **have** $(ss! ?i, v) \in \text{rstep } R$ **unfolding** u **by** *simp*

ultimately show $?thesis$ **by** $(\text{rule } \text{nonroot}'[OF \ \text{len}])$

qed

lemma $\text{rstep-preserves-undefined-root}$:

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

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

proof –

from $\langle \text{wf-trs } R \rangle$ **and** $\langle (\text{Fun } f \ ss, t) \in \text{rstep } R \rangle$ **show** $?thesis$

```

proof (cases rule: rstep-cases-Fun')
  case (root ls r  $\sigma$ )
  then have defined  $R$  ( $f$ , length  $ss$ ) by (auto simp: defined-def)
  with  $\langle \neg \text{defined } R(f, \text{length } ss) \rangle$  show ?thesis by simp
next
  case (nonroot i u) then show ?thesis by simp
qed
qed

```

```

lemma rstep-ctxt-imp-nrrstep: assumes step:  $(s, t) \in \text{rstep } R$  and  $C: C \neq \square$ 
shows  $(C\langle s \rangle, C\langle t \rangle) \in \text{nrrstep } R$ 
proof -
  from step obtain  $l\ r\ D\ \sigma$  where  $(l, r) \in R$   $s = D\langle l \cdot \sigma \rangle$   $t = D\langle r \cdot \sigma \rangle$  by auto
  thus ?thesis unfolding nrrstep-def' using  $C$ 
  by (intro CollectI, unfold split, intro exI[of -  $C \circ_c D$ ] exI conjI, auto) (cases
 $C$ , auto)
qed

```

```

lemma rsteps-ctxt-imp-nrrsteps: assumes steps:  $(s, t) \in (\text{rstep } R)^*$  and  $C: C \neq \square$ 
shows  $(C\langle s \rangle, C\langle t \rangle) \in (\text{nrrstep } R)^*$ 
using steps
proof (induct)
  case (step t u)
  from rstep-ctxt-imp-nrrstep[OF step(2)  $C$ ] step(3) show ?case by simp
qed simp

```

```

lemma nrrstep-mono:
  assumes  $R \subseteq R'$ 
  shows  $\text{nrrstep } R \subseteq \text{nrrstep } R'$ 
  using assms by (force simp: nrrstep-def rstep-r-p-s-def Let-def)

```

```

lemma rrstepE:
  assumes  $(s, t) \in \text{rrstep } R$ 
  obtains  $l$  and  $r$  and  $\sigma$  where  $(l, r) \in R$  and  $s = l \cdot \sigma$  and  $t = r \cdot \sigma$ 
using assms by (auto simp: rrstep-def rstep-r-p-s-def)

```

```

lemma nrrstepE:
  assumes  $(s, t) \in \text{nrrstep } R$ 
  obtains  $C$  and  $l$  and  $r$  and  $\sigma$  where  $C \neq \square$  and  $(l, r) \in R$ 
  and  $s = C\langle l \cdot \sigma \rangle$  and  $t = C\langle r \cdot \sigma \rangle$ 
using assms by (auto simp: nrrstep-def rstep-r-p-s-def)
  (metis ctxt-apply-term.simps(1) poss-Cons-poss subt-at-ctxt-of-pos-term subt-at-nepos-neq)

```

```

lemma singleton-subst-restrict [simp]:
   $\text{subst } x\ s\ |s\ \{x\} = \text{subst } x\ s$ 
  unfolding subst-def subst-restrict-def by (rule ext) simp

```

```

lemma singleton-subst-map [simp]:

```

```

f ∘ subst x s = (f ∘ Var)(x := f s) by (intro ext, auto simp: subst-def)

lemma subst-restrict-vars [simp]:
  (λz. if z ∈ V then f z else g z) |s V = f |s V
unfolding subst-restrict-def
proof (intro ext)
  fix x
  show (if x ∈ V then if x ∈ V then f x else g x else Var x)
    = (if x ∈ V then f x else Var x) by simp
qed

lemma subst-restrict-restrict [simp]:
  assumes V ∩ W = {}
  shows ((λz. if z ∈ V then f z else g z) |s W) = g |s W
unfolding subst-restrict-def
proof (intro ext)
  fix x
  show (if x ∈ W then if x ∈ V then f x else g x else Var x)
    = (if x ∈ W then g x else Var x) using assms by auto
qed

lemma rstep-rstep: rstep (rstep R) = rstep R
proof –
  have ctxt.closure (subst.closure (rstep R)) = rstep R by (simp only: subst-closure-rstep-eq
    ctxt-closure-rstep-eq)
  then show ?thesis unfolding rstep-eq-closure .
qed

lemma rstep-trancl-distrib: rstep (R+) ⊆ (rstep R)+
proof
  fix s t
  assume (s,t) ∈ rstep (R+)
  then show (s,t) ∈ (rstep R)+
  proof
    fix l r C σ
    presume lr: (l,r) ∈ R+ and s: s = C⟨l · σ⟩ and t: t = C⟨r · σ⟩
    from lr have (C⟨l · σ⟩, C⟨r · σ⟩) ∈ (rstep R)+
    proof(induct)
      case (base r)
      then show ?case by auto
    next
      case (step r rr)
      from step(2) have (C⟨r · σ⟩, C⟨rr · σ⟩) ∈ (rstep R) by auto
      with step(3) show ?case by auto
    qed
  then show (s,t) ∈ (rstep R)+ unfolding s t .
  qed auto
qed

```

lemma *rsteps-closed-subst*:
assumes $(s, t) \in (\text{rstep } R)^*$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\text{rstep } R)^*$
using *assms* **and** *subst.closed-rtrancl* [*OF subst-closed-rstep*] **by** (*auto simp: subst.closed-def*)

lemma *join-subst*:
 $\text{subst.closed } r \implies (s, t) \in r^\downarrow \implies (s \cdot \sigma, t \cdot \sigma) \in r^\downarrow$
by (*simp add: join-def subst.closedD subst.closed-comp subst.closed-converse subst.closed-rtrancl*)

lemma *join-subst-rstep* [*intro*]:
 $(s, t) \in (\text{rstep } R)^\downarrow \implies (s \cdot \sigma, t \cdot \sigma) \in (\text{rstep } R)^\downarrow$
by (*intro join-subst, auto*)

lemma *join-ctxt* [*intro*]:
assumes $(s, t) \in (\text{rstep } R)^\downarrow$
shows $(C\langle s \rangle, C\langle t \rangle) \in (\text{rstep } R)^\downarrow$
proof –
from *assms* **obtain** *u* **where** $(s, u) \in (\text{rstep } R)^*$ **and** $(t, u) \in (\text{rstep } R)^*$ **by** *auto*
then have $(C\langle s \rangle, C\langle u \rangle) \in (\text{rstep } R)^*$ **and** $(C\langle t \rangle, C\langle u \rangle) \in (\text{rstep } R)^*$ **by** (*auto intro: rsteps-closed-ctxt*)
then show *?thesis* **by** *blast*
qed

lemma *rstep-simps*:
 $\text{rstep } (R^=) = (\text{rstep } R)^=$
 $\text{rstep } (\text{rstep } R) = \text{rstep } R$
 $\text{rstep } (R \cup S) = \text{rstep } R \cup \text{rstep } S$
 $\text{rstep } \text{Id} = \text{Id}$
 $\text{rstep } (R^{\leftrightarrow}) = (\text{rstep } R)^{\leftrightarrow}$
by *auto*

lemma *rstep-rtrancl-idemp* [*simp*]:
 $\text{rstep } ((\text{rstep } R)^*) = (\text{rstep } R)^*$
proof –
{ **fix** *s t*
assume $(s, t) \in \text{rstep } ((\text{rstep } R)^*)$
then have $(s, t) \in (\text{rstep } R)^*$
by (*induct*) (*metis rsteps-closed-ctxt rsteps-closed-subst*) **}**
then show *?thesis* **by** *auto*
qed

lemma *all-ctxt-closed-rstep-conversion*:
 $\text{all-ctxt-closed UNIV } ((\text{rstep } R)^{\leftrightarrow*})$
unfolding *conversion-def rstep-simps(5)* [*symmetric*] **by** *blast*

definition *instance-rule* :: (*f*, *v*) *rule* \Rightarrow (*f*, *w*) *rule* \Rightarrow *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*) *rule* \Rightarrow (*f*, *v*) *rule* \Rightarrow *bool* **where**
`eq-rule-mod-vars lr st \longleftrightarrow instance-rule lr st \wedge 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 *vars-term* *l*
shows *vars-term* *t* \subseteq *vars-term* *s*
proof –
from *eq*[*unfolded instance-rule-def*]
obtain τ **where** *s*: *s* = *l* · τ **and** *t*: *t* = *r* · τ **by** *auto*
show ?*thesis*
proof
fix *x*
assume *x* \in *vars-term* *t*
from *this*[*unfolded t*] **have** *x* \in *vars-term* (*l* · τ) **using** *vars* **unfolding**
vars-term-subst **by** *auto*
then show *x* \in *vars-term* *s* **unfolding** *s* **by** *auto*
qed
qed

lemma *instance-rule-rstep*: **assumes** *step*: (*s*,*t*) \in *rstep* {*lr*}
and *bex*: *Bex* *R* (*instance-rule* *lr*)
shows (*s*,*t*) \in *rstep* *R*
proof –
from *bex* **obtain** *lr'* **where** *inst*: *instance-rule* *lr lr'* **and** *R*: *lr'* \in *R* **by** *auto*
obtain *l r* **where** *lr*: *lr* = (*l*,*r*) **by** *force*
obtain *l' r'* **where** *lr'*: *lr'* = (*l'*,*r'*) **by** *force*
note *inst* = *inst*[*unfolded lr lr'*]
note *R* = *R*[*unfolded lr'*]
from *inst*[*unfolded instance-rule-def*] **obtain** σ **where** *l*: *l* = *l'* · σ **and** *r*: *r* = *r'* · σ **by** *auto*
from *step*[*unfolded lr*] **obtain** *C* τ **where** *s* = *C* $\langle l \cdot \tau \rangle$ *t* = *C* $\langle r \cdot \tau \rangle$ **by** *auto*
with *l r* **have** *s*: *s* = *C* $\langle l' \cdot (\sigma \circ_s \tau) \rangle$ **and** *t*: *t* = *C* $\langle r' \cdot (\sigma \circ_s \tau) \rangle$ **by** *auto*
from *rstepI*[*OF R s t*] **show** ?*thesis* .
qed

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*
by (*rule* *instance-rule-var-cond*[*OF* - *vars*], *insert eq*[*unfolded eq-rule-mod-vars-def*], *auto*)

lemma *eq-rule-mod-varsE*[*elim*]: **fixes** *l* :: (*f*,*v*)*term*
assumes (*l*,*r*) =_v (*s*,*t*)

```

shows  $\exists \sigma \tau. l = s \cdot \sigma \wedge r = t \cdot \sigma \wedge s = l \cdot \tau \wedge t = r \cdot \tau \wedge \text{range } \sigma \subseteq \text{range } \text{Var} \wedge \text{range } \tau \subseteq \text{range } \text{Var}$ 
proof –
  from assms[unfolded eq-rule-mod-vars-def instance-rule-def fst-conv snd-conv]
  obtain  $\sigma \tau$  where  $l = s \cdot \sigma$  and  $r = t \cdot \sigma$  and  $s = l \cdot \tau$  and  $t = r \cdot \tau$ 
 $\tau$  by blast+
  obtain  $f :: 'f$  where True by auto
  let  $?vst = \text{vars-term } (\text{Fun } f \ [s,t])$ 
  let  $?vtr = \text{vars-term } (\text{Fun } f \ [l,r])$ 
  define  $\sigma'$  where  $\sigma' \equiv \lambda x. \text{if } x \in ?vst \text{ then } \sigma \ x \text{ else } \text{Var } x$ 
  define  $\tau'$  where  $\tau' \equiv \lambda x. \text{if } x \in ?vtr \text{ then } \tau \ x \text{ else } \text{Var } x$ 
  show ?thesis
  proof (intro exI conjI)
    show  $l = s \cdot \sigma'$  unfolding  $l \ \sigma'$ -def
      by (rule term-subst-eq, auto)
    show  $r = t \cdot \sigma'$  unfolding  $r \ \sigma'$ -def
      by (rule term-subst-eq, auto)
    show  $s = l \cdot \tau'$  unfolding  $s \ \tau'$ -def
      by (rule term-subst-eq, auto)
    show  $t = r \cdot \tau'$  unfolding  $t \ \tau'$ -def
      by (rule term-subst-eq, auto)
    have  $\text{Fun } f \ [s,t] \cdot \text{Var} = \text{Fun } f \ [l, r] \cdot \tau'$  unfolding  $s \ t$  by simp
    also have  $\dots = \text{Fun } f \ [s,t] \cdot (\sigma' \circ_s \tau')$  unfolding  $l \ r$  by simp
    finally have  $\text{Fun } f \ [s,t] \cdot (\sigma' \circ_s \tau') = \text{Fun } f \ [s,t] \cdot \text{Var}$  by simp
    from term-subst-eq-rev[OF this] have  $vst: \bigwedge x. x \in ?vst \implies \sigma' \ x \cdot \tau' = \text{Var}$ 
 $x$  unfolding subst-compose-def by auto
    have  $\text{Fun } f \ [l,r] \cdot \text{Var} = \text{Fun } f \ [s, t] \cdot \sigma'$  unfolding  $l \ r$  by simp
    also have  $\dots = \text{Fun } f \ [l,r] \cdot (\tau' \circ_s \sigma')$  unfolding  $s \ t$  by simp
    finally have  $\text{Fun } f \ [l,r] \cdot (\tau' \circ_s \sigma') = \text{Fun } f \ [l,r] \cdot \text{Var}$  by simp
    from term-subst-eq-rev[OF this] have  $vtr: \bigwedge x. x \in ?vtr \implies \tau' \ x \cdot \sigma' = \text{Var } x$ 
unfolding subst-compose-def by auto
    {
      fix  $x$ 
      have  $\sigma' \ x \in \text{range } \text{Var}$ 
      proof (cases  $x \in ?vst$ )
        case True
          from  $vst$ [OF this] show ?thesis by (cases  $\sigma' \ x$ , auto)
        next
          case False
            then show ?thesis unfolding  $\sigma'$ -def by auto
      qed
    }
  then show  $\text{range } \sigma' \subseteq \text{range } \text{Var}$  by auto
  {
    fix  $x$ 
    have  $\tau' \ x \in \text{range } \text{Var}$ 
    proof (cases  $x \in ?vtr$ )
      case True
        from  $vtr$ [OF this] show ?thesis by (cases  $\tau' \ x$ , auto)
      case False
        then show ?thesis unfolding  $\tau'$ -def by auto
    qed
  }

```

```

    next
    case False
    then show ?thesis unfolding  $\tau'$ -def by auto
  qed
}
then show range  $\tau' \subseteq \text{range Var}$  by auto
qed
qed

definition
  linear-trs :: ('f, 'v) trs  $\Rightarrow$  bool
where
  linear-trs R  $\equiv \forall (l, r) \in R. \text{linear-term } l \wedge \text{linear-term } r$ 

lemma linear-trsE[elim, consumes 1]: linear-trs R  $\Longrightarrow (l, r) \in R \Longrightarrow \text{linear-term } l$ 
 $\wedge \text{linear-term } r$ 
  unfolding linear-trs-def by auto

lemma linear-trsI[intro]:  $\llbracket \bigwedge l r. (l, r) \in R \Longrightarrow \text{linear-term } l \wedge \text{linear-term } r \rrbracket \Longrightarrow$ 
linear-trs R
  unfolding linear-trs-def by auto

definition
  left-linear-trs :: ('f, 'v) trs  $\Rightarrow$  bool
where
  left-linear-trs R  $\longleftrightarrow (\forall (l, r) \in R. \text{linear-term } l)$ 

lemma left-linear-trs-union: left-linear-trs (R  $\cup$  S) = (left-linear-trs R  $\wedge$  left-linear-trs
S)
  unfolding left-linear-trs-def by auto

lemma left-linear-mono: assumes left-linear-trs S and R  $\subseteq$  S shows left-linear-trs
R
  using assms unfolding left-linear-trs-def by auto

lemma left-linear-map-funs-trs[simp]: left-linear-trs (map-funs-trs f R) = left-linear-trs
R
  unfolding left-linear-trs-def by (auto simp: map-funs-trs.simps)

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  $\exists t. (s, t) \in \text{rstep } R \wedge \text{weak-match } t v$ 
using weak-match
proof (induct rule: rstep-induct-rule [OF rstep])
  case (1 C sig l r)
  from 1(2) show ?case
  proof (induct C arbitrary: s)

```

```

case (More f bef C aft s)
let ?n = Suc (length bef + length aft)
let ?m = length bef
from More(2) obtain ss where s: s = Fun f ss and lss: ?n = length ss and
wm: (∀ i < length ss. weak-match (ss ! i) ((bef @ C⟨l · sig⟩ # aft) ! i)) by (cases
s, auto)
from lss wm[THEN spec, of ?m] have weak-match (ss ! ?m) C⟨l · sig⟩ by auto
from More(1)[OF this] obtain t where wmt: weak-match t C⟨r · sig⟩ and
step: (ss ! ?m, t) ∈ rstep R by auto
from lss have mss: ?m < length ss by simp
let ?tsi = λ t. take ?m ss @ t # drop (Suc ?m) ss
let ?ts = ?tsi t
let ?ss = ?tsi (ss ! ?m)
from id-take-nth-drop[OF mss]
have lts: length ?ts = ?n using lss by auto
show ?case
proof (rule exI[of - Fun f ?ts], intro conjI)
have weak-match (Fun f ?ts) (More f bef C aft)⟨r · sig⟩ =
weak-match (Fun f ?ts) (Fun f (bef @ C⟨r · sig⟩ # aft)) by simp
also have ... proof (unfold weak-match.simps lts, intro conjI refl allI impI)
fix i
assume i: i < ?n
show weak-match (?ts ! i) ((bef @ C⟨r · sig⟩ # aft) ! i)
proof (cases i = ?m)
case True
have weak-match (?ts ! i) ((bef @ C⟨r · sig⟩ # aft) ! i) = weak-match t
C⟨r · sig⟩
using True mss by (simp add: nth-append)
then show ?thesis using wmt by simp
next
case False
have eq: ?ts ! i = ss ! i ∧ (bef @ C⟨r · sig⟩ # aft) ! i = (bef @ C⟨l · sig⟩
# aft) ! i
proof (cases i < ?m)
case True
then show ?thesis by (simp add: nth-append lss[symmetric])
next
case False
with ⟨i ≠ ?m⟩ i have ∃ j. i = Suc (?m + j) ∧ j < length aft by
presburger
then obtain j where i: i = Suc (?m + j) and j: j < length aft by auto
then have id: (Suc (length bef + j) - min (Suc (length bef + length
aft)) (length bef)) = Suc j by simp
from j show ?thesis by (simp add: nth-append i id lss[symmetric])
qed
then show ?thesis using wm[THEN spec, of i] i[unfolded lss] by (simp)
qed
qed simp
finally show weak-match (Fun f ?ts) (More f bef C aft)⟨r · sig⟩ by simp

```

```

next
  have  $s = \text{Fun } f \text{ ?ss}$  unfolding  $s$  using  $\text{id-take-nth-drop}[OF \text{ mss}, \text{ symmetric}]$ 
by  $\text{simp}$ 
  also have  $\dots = (\text{More } f \text{ (take ?m ss)} \sqcap (\text{drop (Suc ?m) ss})) \langle (ss ! ?m) \rangle$  (is -
 $= ?C \langle \cdot \rangle)$  by  $\text{simp}$ 
  finally have  $s: s = ?C \langle ss ! ?m \rangle$  .
  have  $t: \text{Fun } f \text{ ?ts} = ?C \langle t \rangle$  by  $\text{simp}$ 
  from  $\text{rstep-ctxt}[OF \text{ step}]$ 
  show  $(s, \text{Fun } f \text{ ?ts}) \in \text{rstep } R$ 
  unfolding  $s \ t$  .
qed
next
case  $(\text{Hole } s)$ 
from  $\text{ll } 1(1)$  have  $\text{linear-term } l$  unfolding  $\text{left-linear-trs-def}$  by  $\text{auto}$ 
from  $\text{linear-weak-match}[OF \text{ this Hole[simplified] refl}]$  obtain  $\tau$  where
 $s = l \cdot \tau$  and  $(\forall x \in \text{vars-term } l. \text{ weak-match } (\text{Var } x \cdot \tau) (\text{Var } x \cdot \text{sig}))$ 
by  $\text{auto}$ 
then obtain  $\text{tau}$  where  $s: s = l \cdot \text{tau}$  and  $\text{wm}: (\forall x \in \text{vars-term } l. \text{ weak-match}$ 
 $(\text{tau } x) (\text{Var } x \cdot \text{sig}))$ 
by  $(\text{auto})$ 
let  $\text{?delta} = (\lambda x. \text{ if } x \in \text{vars-term } l \text{ then tau } x \text{ else Var } x \cdot \text{sig})$ 
show  $\text{?case}$ 
proof  $(\text{rule exI}[of - r \cdot \text{?delta}], \text{ rule conjI})$ 
  have  $s = l \cdot (\text{tau } |s (\text{vars-term } l))$  unfolding  $s$  by  $(\text{rule coincidence-lemma})$ 
  also have  $\dots = l \cdot (\text{?delta } |s (\text{vars-term } l))$  by  $\text{simp}$ 
  also have  $\dots = l \cdot \text{?delta}$  by  $(\text{rule coincidence-lemma}[\text{symmetric}])$ 
  finally have  $s: s = l \cdot \text{?delta}$  .
  from  $1(1)$  have  $\text{step}: (l \cdot \text{?delta}, r \cdot \text{?delta}) \in \text{rstep } R$  by  $\text{auto}$ 
  then show  $(s, r \cdot \text{?delta}) \in \text{rstep } R$  unfolding  $s$  .
next
  have  $\text{weak-match } (r \cdot \text{?delta}) (r \cdot \text{sig})$ 
  proof  $(\text{induct } r)$ 
  case  $(\text{Fun } f \text{ ss})$ 
  from  $\text{this}[\text{unfolded set-conv-nth}]$ 
  show  $\text{?case}$  by  $(\text{force})$ 
next
  case  $(\text{Var } x)$ 
  show  $\text{?case}$ 
  proof  $(\text{cases } x \in \text{vars-term } l)$ 
  case  $\text{True}$ 
  with  $\text{wm Var}$  show  $\text{?thesis}$  by  $\text{simp}$ 
next
  case  $\text{False}$ 
  show  $\text{?thesis}$  by  $(\text{simp add: Var False weak-match-refl})$ 
qed
qed
then show  $\text{weak-match } (r \cdot \text{?delta}) (\sqcap \langle (r \cdot \text{sig}) \rangle)$  by  $\text{simp}$ 
qed
qed

```

qed

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* ($rstep\ R$) $\{t\}$

proof

fix f

assume $t0$: $f\ 0 \in \{t\}$ and $chain$: *chain* ($rstep\ R$) f

let $?s = S\ R\ s\ f$

let $?P = \lambda i\ u.\ (?s\ i,\ u) \in rstep\ R \wedge weak-match\ u\ (f\ (Suc\ i))$

have $\forall i.\ (?s\ i,\ ?s\ (Suc\ i)) \in rstep\ R \wedge weak-match\ (?s\ (Suc\ i))\ (f\ (Suc\ i))$

proof

fix i show $(?s\ i,\ ?s\ (Suc\ i)) \in rstep\ R \wedge weak-match\ (?s\ (Suc\ i))\ (f\ (Suc\ i))$

proof (*induct* i)

case 0

from $chain$ have ini : $(f\ 0,\ f\ (Suc\ 0)) \in rstep\ R$ by *simp*

then have $(t,\ f\ (Suc\ 0)) \in rstep\ R$ unfolding *singletonD*[$OF\ t0$, *symmetric*]

.

from *someI-ex*[$OF\ left-linear-weak-match-rstep$ [$OF\ this\ wm\ ll$]]

show $?case$ by *simp*

next

case $(Suc\ i)$

then have $IH1$: $(?s\ i,\ ?s\ (Suc\ i)) \in rstep\ R$

and $IH2$: *weak-match* $(?s\ (Suc\ i))\ (f\ (Suc\ i))$ by *auto*

from $chain$ have nxt : $(f\ (Suc\ i),\ f\ (Suc\ (Suc\ i))) \in rstep\ R$ by *simp*

from *someI-ex*[$OF\ left-linear-weak-match-rstep$ [$OF\ this\ IH2\ ll$]]

have $\exists u.\ ?P\ (Suc\ i)\ u$ by *auto*

from *someI-ex*[$OF\ this$]

show $?case$ by *simp*

qed

qed

moreover have $?s\ 0 = s$ by *simp*

ultimately have $\neg SN-on\ (rstep\ R)\ \{s\}$ by *best*

with SN show *False* by *simp*

qed

end

lemma *lhs-notin-NF-rstep*: $(l,\ r) \in R \implies l \notin NF\ (rstep\ R)$ by *auto*

lemma *NF-instance*:

assumes $(t \cdot \sigma) \in NF \text{ (rstep } R)$ **shows** $t \in NF \text{ (rstep } R)$
using *assms* **by** *auto*

lemma *NF-subterm*:

assumes $t \in NF \text{ (rstep } R)$ **and** $t \sqsupseteq s$
shows $s \in NF \text{ (rstep } R)$
proof (*rule ccontr*)
assume $\neg ?thesis$
then obtain u **where** $(s, u) \in rstep \text{ } R$ **by** *auto*
from $\langle t \sqsupseteq s \rangle$ **obtain** C **where** $t = C\langle s \rangle$ **by** *auto*
with $\langle (s, u) \in rstep \text{ } R \rangle$ **have** $(t, C\langle u \rangle) \in rstep \text{ } R$ **by** *auto*
then have $t \notin NF \text{ (rstep } R)$ **by** *auto*
with *assms* **show** *False* **by** *simp*
qed

abbreviation

$lhss :: ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ terms}$
where
 $lhss \text{ } R \equiv fst \text{ ' } R$

abbreviation

$rhss :: ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ terms}$
where
 $rhss \text{ } R \equiv snd \text{ ' } R$

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

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

lemma *map-funs-term-wa-compose*: $map-funs-term-wa \text{ } gh \text{ } (map-funs-term-wa \text{ } fg \text{ } t) = map-funs-term-wa \text{ } (\lambda(f, n). gh \text{ } (fg \text{ } (f, n), n)) \text{ } t$
by (*induct t, auto*)

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

have $map-funs-trs-wa \text{ } ?fgh \text{ } R = \{(map-funs-term-wa \text{ } ?fgh \text{ } l, map-funs-term-wa \text{ } ?fgh \text{ } r) \mid l \text{ } r. (l, r) \in R\}$ **unfolding** *map-funs-trs-wa-def* **by** *auto*
also have $\dots = \{(map-funs-term-wa \text{ } gh \text{ } (map-funs-term-wa \text{ } fg \text{ } l), map-funs-term-wa \text{ } gh \text{ } (map-funs-term-wa \text{ } fg \text{ } r)) \mid l \text{ } r. (l, r) \in R\}$ **unfolding** *map-funs-term-wa-compose*
..
finally show *?thesis* **unfolding** *map-funs-trs-wa-def* **by** *force*
qed

lemma *map-funs-trs-wa-funas-trs-id*: **assumes** $R: \text{funas-trs } R \subseteq F$
and $\text{id}: \bigwedge g \ n. (g, n) \in F \implies f(g, n) = g$
shows $\text{map-funs-trs-wa } f \ R = R$
proof –
{
 fix $l \ r$
 assume $(l, r) \in R$
 with R **have** $l: \text{funas-term } l \subseteq F$ **and** $r: \text{funas-term } r \subseteq F$ **unfolding** *funas-trs-def*
 by (*force simp: funas-rule-def*) +
 from $\text{map-funs-term-wa-funas-term-id}[OF \ l \ \text{id}] \ \text{map-funs-term-wa-funas-term-id}[OF \ r \ \text{id}]$
 have $\text{map-funs-term-wa } f \ l = l \ \text{map-funs-term-wa } f \ r = r$ **by** *auto*
 } **note** $\text{main} = \text{this}$
 have $\text{map-funs-trs-wa } f \ R = \{(\text{map-funs-term-wa } f \ l, \ \text{map-funs-term-wa } f \ r) \mid l \ r. (l, r) \in R\}$
 unfolding *map-funs-trs-wa-def* **by** *force*
 also have $\dots = R$ **using** main **by** *force*
 finally show *?thesis* .
qed

lemma *map-funs-trs-wa-rstep*: **assumes** $\text{step}: (s, t) \in \text{rstep } R$
shows $(\text{map-funs-term-wa } fg \ s, \text{map-funs-term-wa } fg \ t) \in \text{rstep } (\text{map-funs-trs-wa } fg \ R)$
using *step*
proof (*induct*)
 case ($IH \ C \ \sigma \ l \ r$)
 show *?case* **unfolding** *map-funs-trs-wa-def*
 by (*rule rstepI[where l = map-funs-term-wa fg l and r = map-funs-term-wa fg r and C = map-funs-ctxt-wa fg C], auto simp: IH*)
qed

lemma *map-funs-trs-wa-rsteps*: **assumes** $\text{step}: (s, t) \in (\text{rstep } R)^*$
shows $(\text{map-funs-term-wa } fg \ s, \text{map-funs-term-wa } fg \ t) \in (\text{rstep } (\text{map-funs-trs-wa } fg \ R))^*$
using *step*
proof (*induct*)
 case ($\text{step } a \ b$)
 from $\text{map-funs-trs-wa-rstep}[OF \ \text{step}(2), \text{ of } fg] \ \text{step}(3)$ **show** *?case* **by** *auto*
qed *auto*

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

from $wf\text{-}trs[OF\ IH(1)]\ IH(2)$
show $?case$ **by** $auto$
qed

lemma $rsteps\text{-}ground$:
assumes $wf\text{-}trs: \bigwedge l\ r. (l, r) \in R \implies vars\text{-}term\ r \subseteq vars\text{-}term\ l$
and $ground: ground\ s$
and $steps: (s, t) \in (rstep\ R)^*$
shows $ground\ t$
using $steps\ ground$
by $(induct, insert\ rstep\text{-}ground[OF\ wf\text{-}trs], auto)$

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

lemma $supt\text{-}rstep\text{-}stable$:
assumes $(s, t) \in \{\triangleright\} \cup rstep\ R$
shows $(s \cdot \sigma, t \cdot \sigma) \in \{\triangleright\} \cup rstep\ R$
using $assms$ **proof**
assume $s \triangleright t$ **show** $?thesis$
proof $(rule\ UnI1)$
from $\langle s \triangleright t \rangle$ **show** $s \cdot \sigma \triangleright t \cdot \sigma$ **by** $(rule\ supt\text{-}subst)$
qed
next
assume $(s, t) \in rstep\ R$ **show** $?thesis$
proof $(rule\ UnI2)$
from $\langle (s, t) \in rstep\ R \rangle$ **show** $(s \cdot \sigma, t \cdot \sigma) \in rstep\ R ..$
qed
qed

lemma $supt\text{-}rstep\text{-}transl\text{-}stable$:
assumes $(s, t) \in (\{\triangleright\} \cup rstep\ R)^+$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\{\triangleright\} \cup rstep\ R)^+$
using $assms$ **proof** $(induct)$
case $(base\ u)$
then **have** $(s \cdot \sigma, u \cdot \sigma) \in \{\triangleright\} \cup rstep\ R$ **by** $(rule\ supt\text{-}rstep\text{-}stable)$
then **show** $?case ..$
next
case $(step\ u\ v)$
from $\langle (s \cdot \sigma, u \cdot \sigma) \in (\{\triangleright\} \cup rstep\ R)^+ \rangle$
and $supt\text{-}rstep\text{-}stable[OF\ \langle (u, v) \in \{\triangleright\} \cup rstep\ R \rangle, of\ \sigma]$
show $?case ..$
qed

lemma $supt\text{-}rsteps\text{-}stable$:
assumes $(s, t) \in (\{\triangleright\} \cup rstep\ R)^*$
shows $(s \cdot \sigma, t \cdot \sigma) \in (\{\triangleright\} \cup rstep\ R)^*$
using $assms$
proof $(induct)$

```

    case base then show ?case ..
next
  case (step u v)
  from  $\langle (s, u) \in (\{\triangleright\} \cup \text{rstep } R)^* \rangle$  and  $\langle (u, v) \in \{\triangleright\} \cup \text{rstep } R \rangle$ 
  have  $(s, v) \in (\{\triangleright\} \cup \text{rstep } R)^+$  by (rule rtrancl-into-trancl1)
  from trancl-into-rtrancl[OF supt-rstep-trancl-stable[OF this]]
  show ?case .
qed

lemma eq-rule-mod-vars-refl[simp]:  $r =_v r$ 
proof (cases r)
  case (Pair l r)
  {
    have  $\text{fst } (l, r) = \text{fst } (l, r) \cdot \text{Var} \wedge \text{snd } (l, r) = \text{snd } (l, r) \cdot \text{Var}$  by auto
  }
  then show ?thesis unfolding Pair eq-rule-mod-vars-def instance-rule-def by
best
qed

lemma instance-rule-refl[simp]: instance-rule r r
  using eq-rule-mod-vars-refl[of r] unfolding eq-rule-mod-vars-def by simp

lemma is-Fun-Fun-conv:  $\text{is-Fun } t = (\exists f \text{ ts. } t = \text{Fun } f \text{ ts})$  by auto

lemma wf-trs-def':
  wf-trs R =  $(\forall (l, r) \in R. \text{is-Fun } l \wedge \text{vars-term } r \subseteq \text{vars-term } l)$ 
  by (rule iffI) (auto simp: wf-trs-def is-Fun-Fun-conv)

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

lemma wf-trs-wf-rules[simp]: wf-trs (wf-rules R)
  unfolding wf-trs-def' wf-rules-def wf-rule-def split-def by simp

lemma wf-rules-subset[simp]: wf-rules R  $\subseteq R$ 
  unfolding wf-rules-def by auto

fun wf-reltrs ::  $(f, 'v) \text{ trs} \Rightarrow (f, 'v) \text{ trs} \Rightarrow \text{bool}$  where
  wf-reltrs R S = (
    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
proof (rule ccontr)
  assume  $\neg ?thesis$ 

```

```

then obtain  $l\ r$  where  $\neg \text{wf-trs } R \vee R \neq \{\} \wedge (l,r) \in S \wedge \neg \text{vars-term } r \subseteq$ 
 $\text{vars-term } l$  (is -  $\vee$  ?two) by auto
then show False
proof
  assume  $\neg \text{wf-trs } R$ 
  with  $\text{SN-rstep-imp-wf-trs}[OF \text{ SN-rel-imp-SN}[OF \text{ assms}]]$ 
  show False by simp
next
  assume ?two
  then obtain  $ll\ rr\ x$  where  $lr: (l,r) \in S$  and  $llr: (ll,rr) \in R$  and  $x: x \in$ 
 $\text{vars-term } r$  and  $nx: x \notin \text{vars-term } l$  by auto
  obtain  $f$  and  $\sigma$ 
    where  $\text{sigma}: \sigma = (\lambda y. \text{ if } x = y \text{ then } \text{Fun } f \ [ll,l] \text{ else } \text{Var } y)$  by auto
  have  $\text{id}: \sigma \mid s \ (\text{vars-term } l) = \text{Var}$  unfolding  $\text{sigma}$ 
    by (simp add: subst-restrict-def, rule ext, auto simp: nx)
  have  $l: l = l \cdot \sigma$  by (simp add: coincidence-lemma[of l  $\sigma$ ] id)
  have  $(l \cdot \sigma, r \cdot \sigma) \in \text{rstep } S$  using  $lr$  by auto
  with  $l$  have  $sstep: (l, r \cdot \sigma) \in \text{rstep } S$  by simp
  from  $\text{supteq-subst}[OF \text{ supteq-Var}[OF x], \text{ of } \sigma]$  have
     $r \cdot \sigma \supseteq \text{Fun } f \ [ll,l]$  unfolding  $\text{sigma}$  by auto
  then obtain  $C$  where  $C\langle \text{Fun } f \ [ll, l] \rangle = r \cdot \sigma$  by auto
  with  $sstep$  have  $sstep: (l, C\langle \text{Fun } f \ [ll, l] \rangle) \in \text{rstep } S$  by simp
  obtain  $r$  where  $r: r = \text{relto } (\text{rstep } R) (\text{rstep } S) \cup \{\triangleright\}$  by auto
  have  $(C\langle \text{Fun } f \ [ll,l] \rangle, C\langle \text{Fun } f \ [rr,l] \rangle) \in \text{rstep } R$ 
    by (intro rstepI[OF llrr, of - C  $\circ_c$  More f [] [] [l] Var], auto)
  with  $sstep$  have  $\text{relto}: (l, C\langle \text{Fun } f \ [rr,l] \rangle) \in r$  unfolding  $r$  by auto
  have  $C\langle \text{Fun } f \ [rr,l] \rangle \supseteq \text{Fun } f \ [rr,l]$  using  $\text{ctxt-imp-supteq}$  by auto
  also have  $\text{Fun } f \ [rr,l] \triangleright l$  by auto
  finally have  $\text{supt}: C\langle \text{Fun } f \ [rr,l] \rangle \triangleright l$  unfolding  $\text{supt-def}$  by simp
  then have  $(C\langle \text{Fun } f \ [rr,l] \rangle, l) \in r$  unfolding  $r$  by auto
  with  $\text{relto}$  have  $\text{loop}: (l, l) \in r^+$  by auto
  have  $\text{SN } r$  unfolding  $r$ 
    by (rule SN-imp-SN-union-supt[OF SN-rel[unfolded SN-rel-defs]], blast)
  then have  $\text{SN } (r^+)$  by (rule SN-imp-SN-trancl)
  with  $\text{loop}$  show False unfolding  $\text{SN-on-def}$  by auto
qed
qed

```

lemmas $\text{rstep-wf-rules-subset} = \text{rstep-mono}[OF \text{ wf-rules-subset}]$

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

lemma $\text{map-vars-trs-rstep}$:
assumes $(s, t) \in \text{rstep } (\text{map-vars-trs } f \ R)$ **(is - $\in \text{rstep } ?R$)**
shows $(s \cdot \tau, t \cdot \tau) \in \text{rstep } R$
using assms
proof
fix $ml\ mr\ C\ \sigma$

presume $mem: (ml, mr) \in ?R$ **and** $s: s = C\langle ml \cdot \sigma \rangle$ **and** $t: t = C\langle mr \cdot \sigma \rangle$
let $?m = map\text{-}vars\text{-}term\ f$
from mem **obtain** $l\ r$ **where** $mem: (l, r) \in R$ **and** $id: ml = ?m\ l\ mr = ?m\ r$
unfolding $map\text{-}vars\text{-}trs\text{-}def$ **by** $auto$
have $id: s \cdot \tau = (C \cdot_c \tau)\langle ?m\ l \cdot \sigma \circ_s \tau \rangle\ t \cdot \tau = (C \cdot_c \tau)\langle ?m\ r \cdot \sigma \circ_s \tau \rangle$ **by** $(auto\ simp: s\ t\ id)$
then show $(s \cdot \tau, t \cdot \tau) \in rstep\ R$
unfolding $id\ apply\text{-}subst\text{-}map\text{-}vars\text{-}term$
using mem **by** $auto$
qed $auto$

lemma $map\text{-}vars\text{-}rsteps$:
assumes $(s, t) \in (rstep\ (map\text{-}vars\text{-}trs\ f\ R))^*$ **(is** $- \in (rstep\ ?R)^*$ **)**
shows $(s \cdot \tau, t \cdot \tau) \in (rstep\ R)^*$
using $assms$
proof $(induct)$
case $base$ **then show** $?case$ **by** $simp$
next
case $(step\ t\ u)$
from $map\text{-}vars\text{-}trs\text{-}rstep[OF\ step(2),\ of\ \tau]\ step(3)$ **show** $?case$ **by** $auto$
qed

lemma $rsteps\text{-}subst\text{-}closed$: $(s, t) \in (rstep\ R)^+ \implies (s \cdot \sigma, t \cdot \sigma) \in (rstep\ R)^+$
proof $-$
let $?R = rstep\ R$
assume $steps: (s, t) \in ?R^+$
have $subst: subst.closed\ (?R^+)$ **by** $(rule\ subst.closed\ trancl[OF\ subst.closed\ rstep])$
from $this[unfolded\ subst.closed\ def]\ steps$ **show** $?thesis$ **by** $auto$
qed

lemma $supteq\text{-}rtrancl\text{-}supt$:
 $(R^+ \ O\ \{\triangleright\}) \subseteq (\{\triangleright\} \cup R)^+$ **(is** $?l \subseteq ?r$ **)**
proof
fix $x\ z$
assume $(x, z) \in ?l$
then obtain y **where** $xy: (x, y) \in R^+$ **and** $yz: y \triangleright z$ **by** $auto$
from xy **have** $xy: (x, y) \in ?r$ **by** $(rule\ trancl\text{-}mono,\ simp)$
show $(x, z) \in ?r$
proof $(cases\ y = z)$
case $True$
with xy **show** $?thesis$ **by** $simp$
next
case $False$
with yz **have** $yz: (y, z) \in \{\triangleright\} \cup R$ **by** $auto$
with xy **have** $xz: (x, z) \in ?r\ O\ (\{\triangleright\} \cup R)$ **by** $auto$
then show $?thesis$ **by** $(metis\ UnCI\ trancl\ unfold)$
qed
qed

lemma *rrstepI*[intro]: $(l, r) \in R \implies s = l \cdot \sigma \implies t = r \cdot \sigma \implies (s, t) \in \text{rrstep } R$
unfolding *rrstep-def'* **by** *auto*

lemma *CS-rrstep-conv*: *subst.closure* = *rrstep*
apply (*intro ext*)
apply (*unfold rrstep-def'*)
apply (*intro subset-antisym*)
by (*insert subst.closure.cases, blast, auto*)

3.5 Rewrite steps at a fixed position

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

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

lemma *rstep-pos-subst*:
assumes $(s, t) \in \text{rstep-pos } R \text{ } p$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{rstep-pos } R \text{ } p$
using *assms*
proof (*cases*)
case (*rule l r σ*)
with *rstep-pos.intros* [*OF this*(2), *of p s · σ σ' ◦_s σ*]
show ?thesis **by** (*auto simp: ctxt-of-pos-term-subst*)
qed

lemma *rstep-pos-rule*:
assumes $(l, r) \in R$
shows $(l, r) \in \text{rstep-pos } R \text{ } []$
using *rstep-pos.intros* [*OF assms, of [] l Var*] **by** *simp*

lemma *rstep-pos-rstep-r-p-s-conv*:
 $\text{rstep-pos } R \text{ } p = \{(s, t) \mid s \text{ } t \text{ } r \text{ } \sigma. (s, t) \in \text{rstep-r-p-s } R \text{ } p \text{ } \sigma\}$
by (*auto simp: rstep-r-p-s-def Let-def subt-at-ctxt-of-pos-term*
intro: replace-at-ident
elim!: rstep-pos.cases)

lemma *rstep-rstep-pos-conv*:
 $\text{rstep } R = \{(s, t) \mid s \text{ } t \text{ } p. (s, t) \in \text{rstep-pos } R \text{ } p\}$
by (*force simp: rstep-pos-rstep-r-p-s-conv rstep-iff-rstep-r-p-s*)

lemma *rstep-pos-supt*:
assumes $(s, t) \in \text{rstep-pos } R \text{ } p$
and $q: q \in \text{poss } u$ **and** $u: u \mid\!-\! q = s$
shows $(u, (\text{ctxt-of-pos-term } q \text{ } u)(t)) \in \text{rstep-pos } R \text{ } (q @ p)$
using *assms*
proof (*cases*)
case (*rule l r σ*)
with *q* **and** *u* **have** $(q @ p) \in \text{poss } u$ **and** $u \mid\!-\! (q @ p) = l \cdot \sigma$ **by** *auto*

with *rstep-pos.rule* [*OF rule(2) this*] **show** *?thesis*
unfolding rule by (*auto simp: ctxt-of-pos-term-append u*)
qed

lemma *rrstep-rstep-pos-conv*:
rrstep R = rstep-pos R []
by (*auto simp: rrstep-def rstep-pos-rstep-r-p-s-conv*)

lemma *rrstep-imp-rstep*:
assumes $(s, t) \in \text{rrstep } R$
shows $(s, t) \in \text{rstep } R$
using *assms* **by** (*auto simp: rrstep-def rstep-iff-rstep-r-p-s*)

lemma *not-NF-rstep-imp-subteq-not-NF-rrstep*:
assumes $s \notin \text{NF } (\text{rstep } R)$
shows $\exists t \trianglelefteq s. t \notin \text{NF } (\text{rrstep } R)$
proof –
from *assms* **obtain** *u* **where** $(s, u) \in \text{rstep } R$ **by** *auto*
then obtain $l \ r \ C \ \sigma$ **where** $(l, r) \in R$ **and** $s = C\langle l \cdot \sigma \rangle$ **and** $u = C\langle r \cdot \sigma \rangle$ **by** *auto*
then have $(l \cdot \sigma, r \cdot \sigma) \in \text{rrstep } R$ **and** $l \cdot \sigma \trianglelefteq s$ **by** *auto*
then show *?thesis* **by** *blast*
qed

lemma *all-subt-NF-rrstep-iff-all-subt-NF-rstep*:
 $(\forall s \triangleleft t. s \in \text{NF } (\text{rrstep } R)) \longleftrightarrow (\forall s \triangleleft t. s \in \text{NF } (\text{rstep } R))$
by (*auto dest: rrstep-imp-rstep supt-supteq-trans not-NF-rstep-imp-subteq-not-NF-rrstep*)

lemma *not-in-poss-imp-NF-rstep-pos* [*simp*]:
assumes $p \notin \text{poss } s$
shows $s \in \text{NF } (\text{rstep-pos } R \ p)$
using *assms* **by** (*auto simp: NF-def elim: rstep-pos.cases*)

lemma *Var-rstep-imp-rstep-pos-Empty*:
assumes $(\text{Var } x, t) \in \text{rstep } R$
shows $(\text{Var } x, t) \in \text{rstep-pos } R []$
using *assms* **by** (*metis Var-supt nrrstep-subt rrstep-rstep-pos-conv rstep-cases*)

lemma *rstep-args-NF-imp-rrstep*:
assumes $(s, t) \in \text{rstep } R$
and $\forall u \triangleleft s. u \in \text{NF } (\text{rstep } R)$
shows $(s, t) \in \text{rrstep } R$
using *assms* **by** (*metis NF-iff-no-step nrrstep-subt rstep-cases*)

lemma *rstep-pos-imp-rstep-pos-Empty*:
assumes $(s, t) \in \text{rstep-pos } R \ p$
shows $(s \mid - \ p, t \mid - \ p) \in \text{rstep-pos } R []$
using *assms* **by** (*cases*) (*auto simp: replace-at-subt-at intro: rstep-pos-rule rstep-pos-subst*)

```

lemma rstep-pos-arg:
  assumes  $(s, t) \in \text{rstep-pos } R \ p$ 
  and  $i < \text{length } ss$  and  $ss ! i = s$ 
  shows  $(\text{Fun } f \ ss, (\text{ctxt-of-pos-term } [i] (\text{Fun } f \ ss))\langle t \rangle) \in \text{rstep-pos } R \ (i \# p)$ 
using assms
apply (cases)
  by auto (metis term.sel(4) ctxt-apply-term.simps(2) ctxt-of-pos-term.simps(2)
poss-Cons-poss rstep-pos.intros subt-at.simps(2))

lemma rstep-imp-max-pos:
  assumes  $(s, t) \in \text{rstep } R$ 
  shows  $\exists u. \exists p \in \text{poss } s. (s, u) \in \text{rstep-pos } R \ p \wedge (\forall v \triangleleft s \mid - p. v \in \text{NF } (\text{rstep } R))$ 
using assms
proof (induction s arbitrary: t)
  case (Var x)
    from Var-rstep-imp-rstep-pos-Empty [OF this] show ?case by auto
  next
    case (Fun f ss)
    show ?case
    proof (cases  $\forall v \triangleleft \text{Fun } f \ ss \mid - \square. v \in \text{NF } (\text{rstep } R)$ )
      case True
        moreover with Fun.prems
          have  $(\text{Fun } f \ ss, t) \in \text{rstep-pos } R \ \square$ 
          by (auto dest: rstep-args-NF-imp-rrstep simp: rrstep-rstep-pos-conv)
          ultimately show ?thesis by auto
      next
        case False
        then obtain v where  $v \triangleleft \text{Fun } f \ ss$  and  $v \notin \text{NF } (\text{rstep } R)$  by auto
        then obtain s and w where  $s \in \text{set } ss$  and  $s \triangleright v$  and  $(s, w) \in \text{rstep } R$ 
        by (auto simp: NF-def) (metis NF-iff-no-step NF-subterm supt-Fun-imp-arg-supteq)
        from Fun.IH [OF this(1, 3)] obtain u and p
          where  $p \in \text{poss } s$  and  $*(s, u) \in \text{rstep-pos } R \ p$ 
          and  $** : \forall v \triangleleft s \mid - p. v \in \text{NF } (\text{rstep } R)$  by blast
        from  $\langle s \in \text{set } ss \rangle$  obtain i
          where  $i < \text{length } ss$  and  $[simp]: ss ! i = s$  by (auto simp: in-set-conv-nth)
        with  $\langle p \in \text{poss } s \rangle$  have  $i \# p \in \text{poss } (\text{Fun } f \ ss)$  by auto
        moreover with  $**$  have  $\forall v \triangleleft \text{Fun } f \ ss \mid - (i \# p). v \in \text{NF } (\text{rstep } R)$  by auto
        moreover from rstep-pos-arg [OF  $* \langle i < \text{length } ss \rangle \langle ss ! i = s \rangle$ ]
          have  $(\text{Fun } f \ ss, (\text{ctxt-of-pos-term } [i] (\text{Fun } f \ ss))\langle u \rangle) \in \text{rstep-pos } R \ (i \# p)$  .
          ultimately show ?thesis by blast
    qed
  qed

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  $\neg \text{SN } (\text{rstep } R)$ 
  using rhs-free-vars-imp-rstep-not-SN [OF assms] by (auto simp: SN-defs)

lemma SN-imp-variable-condition:

```

assumes $SN \ (rstep \ R)$
shows $\forall (l, r) \in R. \text{vars-term } r \subseteq \text{vars-term } l$
using *assms* **and** *rhs-free-vars-imp-rstep-not-SN'* [of - - R] **by** *blast*

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

3.6 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]: $\llbracket \bigwedge i. i < \text{length } ts \Longrightarrow (ss ! i, ts ! i) \in \text{par-rstep } R \rrbracket \Longrightarrow$
 $\text{length } ss = \text{length } ts$
 $\Longrightarrow (\text{Fun } f \ ss, \text{Fun } f \ ts) \in \text{par-rstep } R$
| *par-step-var*[intro]: $(\text{Var } x, \text{Var } x) \in \text{par-rstep } R$

lemma *par-rstep-refl*[intro]: $(t, t) \in \text{par-rstep } R$
by (*induct t, auto*)

lemma *all-ctxt-closed-par-rstep*[intro]: *all-ctxt-closed* $F \ (\text{par-rstep } R)$
unfolding *all-ctxt-closed-def*
by *auto*

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

proof(*induct n arbitrary:ys*)

case 0

then have $\forall i < \text{length } xs. (xs ! i = ys ! i)$ **by** *simp*
with 0 **show** ?case **using** *relpow-0-I list-eq-iff-nth-eq* **by** *metis*

next

case (*Suc n*)

let ?c = $\lambda z i. (xs ! i, z) \in \text{par-rstep } R \rightsquigarrow n \wedge (z, ys ! i) \in \text{par-rstep } R$

{ **fix** i **assume** $i < \text{length } xs$

from *relpow-Suc-E*[*OF Suc*(\mathcal{I})](*rule-format*, *OF this*)

have $\exists z. (?c \ z \ i)$ **by** *metis*

}

with choice **have** $\exists zf. \forall i < \text{length } xs. (?c \ (zf \ i) \ i)$ **by** *meson*

then obtain zf **where** $a: \forall i < \text{length } xs. (?c \ (zf \ i) \ i)$ **by** *auto*

let ?zs = $\text{map } zf \ [0..<\text{length } xs]$

have $\text{len}:\text{length } xs = \text{length } ?zs$ **by** *simp*

from a *map-nth* **have** $\forall i < \text{length } xs. (xs ! i, ?zs ! i) \in \text{par-rstep } R \rightsquigarrow n$ **by**

auto

from *Suc*(1)[*OF len this*] **have** $n:(\text{Fun } f \ xs, \text{Fun } f \ ?zs) \in \text{par-rstep } R \rightsquigarrow n$ **by**

auto

from a *map-nth* **have** $\forall i < \text{length } xs. (?zs ! i, ys ! i) \in \text{par-rstep } R$ **by** *auto*

with *par-step-fun len Suc*(2) **have** $(\text{Fun } f \ ?zs, \text{Fun } f \ ys) \in \text{par-rstep } R$ **by** *auto*

with n **show** ?case **by** *auto*

qed

```

lemma ctxt-closed-par-rstep[intro]: ctxt.closed (par-rstep R)
proof (rule one-imp-ctxt-closed)
  fix f bef s t aft
  assume st: (s,t) ∈ par-rstep R
  let ?ss = bef @ s # aft
  let ?ts = bef @ t # aft
  show (Fun f ?ss, Fun f ?ts) ∈ par-rstep R
  proof (rule par-step-fun)
    fix i
    assume i < length ?ts
    show (?ss ! i, ?ts ! i) ∈ par-rstep R
      using par-rstep-refl[of ?ts ! i R] st by (cases i = length bef, auto simp:
nth-append)
    qed simp
  qed

lemma subst-closed-par-rstep: (s,t) ∈ par-rstep R ⇒ (s · σ, t · σ) ∈ par-rstep R
proof (induct rule: par-rstep.induct)
  case (root-step s t τ)
  show ?case
    using par-rstep.root-step[OF root-step, of τ ∘s σ] by auto
next
  case (par-step-var x)
  show ?case by auto
next
  case (par-step-fun ss ts f)
  show ?case unfolding eval-term.simps
    by (rule par-rstep.par-step-fun, insert par-step-fun(2-3), auto)
qed

lemma R-par-rstep: R ⊆ par-rstep R
  using root-step[of - - R Var] by auto

lemma par-rstep-rsteps: par-rstep R ⊆ (rstep R)*
proof
  fix s t
  assume (s,t) ∈ par-rstep R
  then show (s,t) ∈ (rstep R)*
  proof (induct rule: par-rstep.induct)
    case (root-step s t sigma)
    then show ?case by auto
  next
    case (par-step-var x)
    then show ?case by auto
  next
    case (par-step-fun ts ss f)
    from all-ctxt-closedD[of UNIV, OF all-ctxt-closed-rsteps - par-step-fun(3)]

```

```

par-step-fun(2)]
  show ?case unfolding par-step-fun(3) by simp
qed
qed

lemma rstep-par-rstep: rstep R ⊆ par-rstep R
  by (rule rstep-subset[OF ctxt-closed-par-rstep subst.closedI R-par-rstep],
      insert subst-closed-par-rstep, auto)

lemma par-rsteps-rsteps: (par-rstep R)* = (rstep R)* (is ?P = ?R)
proof
  from rtrancl-mono[OF par-rstep-rsteps[of R]] show ?P ⊆ ?R by simp
  from rtrancl-mono[OF rstep-par-rstep] show ?R ⊆ ?P .
qed

lemma par-rsteps-union: (par-rstep A ∪ par-rstep B)* =
  (rstep (A ∪ B))*
proof
  show (par-rstep A ∪ par-rstep B)* ⊆ (rstep (A ∪ B))*
    by (metis par-rsteps-rsteps rstep-union rtrancl-Un-rtrancl set-eq-subset)
  show (rstep (A ∪ B))* ⊆ (par-rstep A ∪ par-rstep B)* unfolding rstep-union
    by (meson rstep-par-rstep rtrancl-mono sup-mono)
qed

lemma par-rstep-inverse: par-rstep (R^-1) = (par-rstep R)^-1
proof -
  {
    fix s t :: ('a,'b)term and R
    assume (s,t) ∈ par-rstep (R^-1)
    hence (t,s) ∈ par-rstep R
      by (induct s t, auto)
  }
  from this[of - - R] this[of - - R^-1]
  show ?thesis by auto
qed

lemma par-rstep-conversion: (rstep R)↔* = (par-rstep R)↔*
  unfolding conversion-def
  by (metis par-rsteps-rsteps rtrancl-Un-rtrancl rtrancl-converse)

lemma par-rstep-mono: assumes R ⊆ S
  shows par-rstep R ⊆ par-rstep S
proof
  fix s t
  show (s, t) ∈ par-rstep R ⟹ (s, t) ∈ par-rstep S
    by (induct s t rule: par-rstep.induct, insert assms, auto)
qed

lemma wf-trs-par-rstep: assumes wf: ⋀ l r. (l,r) ∈ R ⟹ is-Fun l

```

```

and step: (Var x, t) ∈ par-rstep R
shows t = Var x
using step
proof (cases rule: par-rstep.cases)
  case (root-step l r σ)
    from root-step(1) wf[OF root-step(3)] show ?thesis by (cases l, auto)
qed auto

```

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 lin: linear-term l
  and step: (l · σ, t) ∈ par-rstep R
  shows ( $\exists \tau. t = l \cdot \tau \wedge (\forall x \in \text{vars-term } l. (\sigma x, \tau x) \in \text{par-rstep } R) \vee$ 
    ( $\exists C l'' l' r'. l = C \langle l'' \rangle \wedge \text{is-Fun } l'' \wedge (l', r') \in R \wedge (l'' \cdot \sigma = l' \cdot \tau) \wedge$ 
    ( $(C \cdot_c \sigma) \langle r' \cdot \tau \rangle, t) \in \text{par-rstep } R$ ))
  using lin step
proof (induction l arbitrary: t)
  case (Var x t)
    let ?tau =  $\lambda y. t$ 
    show ?case
    by (rule exI[of - ?tau], rule disjI1, insert Var(2), auto)
next
  case (Fun f ss)
    let ?ss = map ( $\lambda s. s \cdot \sigma$ ) ss
    let ?R = par-rstep R
    from Fun(3)
    show ?case
    proof (cases rule: par-rstep.cases)
      case (root-step l r τ)
        show ?thesis
        proof (rule exI, rule disjI2, intro exI conjI)
          show (l, r) ∈ R by (rule root-step(3))
          show Fun f ss =  $\square \langle \text{Fun } f \text{ ss} \rangle$  by simp
          show (Fun f ss) · σ = l · τ by (rule root-step(1))
          show ( $(\square \cdot_c \sigma) \langle r \cdot \tau \rangle, t$ ) ∈ ?R unfolding root-step(2) using par-rstep-refl
by simp
        qed simp
      next
      case (par-step-var x)
        then show ?thesis by simp
      next
      case (par-step-fun ts ss1 g)
        then have id: ss1 = ?ss g = f and len: length ts = length ss by auto
        let ?p1 =  $\lambda \tau i. ts ! i = ss ! i \cdot \tau \wedge (\forall x \in \text{vars-term } (ss ! i). (\sigma x, \tau x) \in ?R)$ 
        let ?p2 =  $\lambda \tau i. (\exists C l'' l' r'. ss ! i = C \langle l'' \rangle \wedge \text{is-Fun } l'' \wedge (l', r') \in R \wedge l'' \cdot \sigma = l' \cdot \tau \wedge ((C \cdot_c \sigma) \langle r' \cdot \tau \rangle, (ts ! i)) \in ?R)$ 
        let ?p =  $\lambda \tau i. ?p1 \ \tau \ i \vee ?p2 \ \tau \ i$ 
        {
          fix i

```

```

    assume  $i: i < \text{length } ss$ 
    with  $\text{par-step-fun}(4)$   $id$  have  $i2: i < \text{length } ts$  by auto
    from  $\text{par-step-fun}(3)[OF\ i2]$  have  $\text{step}: (ss ! i \cdot \sigma, ts ! i) \in \text{par-rstep } R$ 
  unfolding  $id\ \text{nth-map}[OF\ i]$  .
    from  $i$  have  $\text{mem}: ss ! i \in \text{set } ss$  by auto
    from  $\text{Fun.prem}(1)\ \text{mem}$  have  $\text{linear-term } (ss ! i)$  by auto
    from  $\text{Fun.IH}[OF\ \text{mem}\ \text{this}\ \text{step}]$  have  $\exists\ \tau. ?p\ \tau\ i$  .
  }
  then have  $\forall\ i. \exists\ \tau. i < \text{length } ss \longrightarrow ?p\ \tau\ i$  by blast
  from  $\text{choice}[OF\ \text{this}]$  obtain  $\text{taus}$  where  $\text{taus}: \bigwedge i. i < \text{length } ss \implies ?p\ (\text{taus}\ i)$ 
  by blast
  show ?thesis
  proof (cases  $\exists\ i. i < \text{length } ss \wedge ?p2\ (\text{taus}\ i)\ i$ )
    case True
      then obtain  $i$  where  $i: i < \text{length } ss$  and  $p2: ?p2\ (\text{taus}\ i)\ i$  by blast
      from  $\text{par-step-fun}(2)[\text{unfolded}\ id]$  have  $t: t = \text{Fun } f\ ts$  .
      from  $i$  have  $i': i' < \text{length } ts$  unfolding  $\text{len}$  .
      from  $p2$  obtain  $C\ l''\ l'\ r'$  where  $\text{ssi}: ss ! i = C\ \langle l'' \rangle$  and  $\text{is-Fun } l''\ (l', r')$ 
       $\in R\ l'' \cdot \sigma = l' \cdot \text{taus}\ i$ 
      and  $\text{tsi}: ((C \cdot_c \sigma) \langle r' \cdot \text{taus}\ i \rangle, ts ! i) \in ?R$  by blast
      from  $\text{id-take-nth-drop}[OF\ i, \text{unfolded}\ \text{ssi}]$  obtain  $\text{bef}\ \text{aft}$  where  $\text{ss}: ss = \text{bef}$ 
    @  $C\ \langle l'' \rangle \# \text{aft}$ 
      and  $\text{bef}: \text{bef} = \text{take } i\ ss$ 
      and  $\text{aft}: \text{aft} = \text{drop } (\text{Suc } i)\ ss$  by blast
      let  $?C = \text{More } f\ \text{bef}\ C\ \text{aft}$ 
      let  $?r = (C \cdot_c \sigma) \langle r' \cdot \text{taus}\ i \rangle$ 
      let  $?sig = \text{map } (\lambda s. s \cdot \sigma)$ 
      let  $?bra = ?sig\ \text{bef}\ @\ ?r\ \# ?sig\ \text{aft}$ 
      have  $C: (?C \cdot_c \sigma) \langle r' \cdot \text{taus}\ i \rangle = \text{Fun } f\ ?bra$  by simp
      show ?thesis unfolding  $\text{ss}$ 
      proof (rule  $\text{exI}[\text{of } -\ \text{taus}\ i]$ , rule  $\text{disjI2}$ , rule  $\text{exI}[\text{of } -\ ?C]$ , intro  $\text{exI}\ \text{conjI}$ )
        show  $\text{is-Fun } l''$  by fact
        show  $(l', r') \in R$  by fact
        show  $l'' \cdot \sigma = l' \cdot \text{taus}\ i$  by fact
        show  $((?C \cdot_c \sigma) \langle r' \cdot \text{taus}\ i \rangle, t) \in ?R$  unfolding  $C\ t$ 
        proof (rule  $\text{par-rstep.par-step-fun}$ )
          show  $\text{length } ?bra = \text{length } ts$ 
          unfolding  $\text{len}\ \text{unfolding}\ ss$  by simp
        next
          fix  $j$ 
          assume  $j: j < \text{length } ts$ 
          show  $(?bra ! j, ts ! j) \in ?R$ 
          proof (cases  $j = i$ )
            case True
              then have  $?bra ! j = ?r$  using  $\text{bef}\ i$  by (simp add: nth-append)
              then show ?thesis using  $\text{tsi}\ \text{True}$  by simp
            next
              case False
              from  $\text{bef}\ i$  have  $\min(\text{length } ss)\ i = i$  by simp

```

```

      then have ?bra ! j = (?sig bef @ (C < l'' > · σ) # ?sig aft) ! j using
False bef i by (simp add: nth-append)
      also have ... = ?sig ss ! j unfolding ss by simp
      also have ... = ss1 ! j unfolding id ..
      finally show ?thesis
      using par-step-fun(3)[OF j] by auto
    qed
  qed
qed simp
next
case False
with taus have taus:  $\bigwedge i. i < \text{length } ss \implies ?p1 \text{ (taus } i) i$  by blast
from Fun(2) have is-partition (map vars-term ss) by simp
from subst-merge[OF this, of taus] obtain τ where tau:  $\bigwedge i x. i < \text{length } ss$ 
 $\implies x \in \text{vars-term } (ss ! i) \implies \tau x = \text{taus } i x$  by auto
let ?tau = τ
{
  fix i
  assume i:  $i < \text{length } ss$ 
  then have mem:  $ss ! i \in \text{set } ss$  by auto
  from taus[OF i] have p1: ?p1 (taus i) i .
  have id:  $ss ! i \cdot (\text{taus } i) = ss ! i \cdot \tau$ 
    by (rule term-subst-eq, rule tau[OF i, symmetric])
  have ?p1 ?tau i
  proof (rule conjI[OF - ballI])
    fix x
    assume x:  $x \in \text{vars-term } (ss ! i)$ 
    with p1 have step:  $(\sigma x, \text{taus } i x) \in \text{par-rstep } R$  by auto
    with tau[OF i x]
    show  $(\sigma x, ?tau x) \in \text{par-rstep } R$  by simp
  qed (insert p1[unfolded id], auto)
} note p1 = this
have p1:  $\bigwedge i. i < \text{length } ss \implies ?p1 \text{ } \tau i$  by (rule p1)
let ?ss = map ( $\lambda s. s \cdot \tau$ ) ss
show ?thesis unfolding par-step-fun(2) id
proof (rule exI[of - τ], rule disjI1, rule conjI[OF - ballI])
  have ts = map ( $\lambda i. ts ! i$ ) [0 ..< (length ts)] by (rule map-nth[symmetric])
  also have ... = map ( $\lambda i. ?ss ! i$ ) [0 ..< length ?ss] unfolding len using
p1 by auto
  also have ... = ?ss by (rule map-nth)
  finally have ts:  $ts = ?ss$  .
  show Fun f ts = Fun f ss · τ unfolding ts by auto
next
fix x
assume x  $x \in \text{vars-term } (\text{Fun } f ss)$ 
then obtain s where s:  $s \in \text{set } ss$  and x:  $x \in \text{vars-term } s$  by auto
from s[unfolded set-conv-nth] obtain i where i:  $i < \text{length } ss$  and s:  $s =$ 
ss ! i by auto
from p1[OF i] x[unfolded s]

```

```

      show  $(\sigma\ x, \tau\ x) \in \text{par-rstep}\ R$  by blast
    qed
  qed
qed

```

lemma *par-rstep-id*:
 $(s, t) \in R \implies (s, t) \in \text{par-rstep}\ R$
 using *par-rstep.root-step* [of *s t R Var*] by *simp*

3.7 Function Symbols and Variables

3.8 Function Symbols and Variables

fun *root-list* :: $('f, 'v)\ \text{term} \Rightarrow ('f \times \text{nat})\ \text{list}$

where

```

  root-list (Var x) = [] |
  root-list (Fun f ts) = [(f, length ts)]

```

definition *vars-rule-list* :: $('f, 'v)\ \text{rule} \Rightarrow 'v\ \text{list}$

where

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

definition *fun-rule-list* :: $('f, 'v)\ \text{rule} \Rightarrow 'f\ \text{list}$

where

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

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

where

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

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

where

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

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

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*

by (*simp* add: *vars-rule-list-def vars-rule-def*)

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

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

by (*simp* add: *fun-rule-list-def fun-rule-def*)

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

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

by (*simp* add: *funas-rule-list-def funas-rule-def*)

lemma *set-roots-rule-list* [*simp*]:
 $set (roots-rule-list\ r) = roots-rule\ r$
by (*cases fst r snd r rule: term.exhaust [case-product term.exhaust]*)
(auto simp: roots-rule-list-def roots-rule-def ac-simps)

lemma *set-funas-args-rule-list* [*simp*]:
 $set (funas-args-rule-list\ r) = funas-args-rule\ r$
by (*simp add: funas-args-rule-list-def funas-args-rule-def*)

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*]:
 $set (vars-trs-list\ trs) = vars-trs\ (set\ trs)$
by (*simp add: vars-trs-def vars-trs-list-def*)

lemma *set-funs-trs-list* [*simp*]:
 $set (funs-trs-list\ R) = funs-trs\ (set\ R)$
by (*simp add: funs-trs-def funs-trs-list-def*)

lemma *set-funas-trs-list* [*simp*]:
 $set (funas-trs-list\ R) = funas-trs\ (set\ R)$
by (*simp add: funas-trs-def funas-trs-list-def*)

lemma *set-roots-trs-list* [*simp*]:
 $set (roots-trs-list\ R) = roots-trs\ (set\ R)$
by (*simp add: roots-trs-def roots-trs-list-def*)

lemma *set-funas-args-trs-list* [*simp*]:
 $set (funas-args-trs-list\ R) = funas-args-trs\ (set\ R)$
by (*simp add: funas-args-trs-def funas-args-trs-list-def*)

lemmas $\text{vars-list-defs} = \text{vars-trs-list-def vars-rule-list-def}$
lemmas $\text{funas-list-defs} = \text{funas-trs-list-def funas-rule-list-def}$
lemmas $\text{funas-list-defs} = \text{funas-trs-list-def funas-rule-list-def}$
lemmas $\text{roots-list-defs} = \text{roots-trs-list-def roots-rule-list-def}$
lemmas $\text{funas-args-list-defs} = \text{funas-args-trs-list-def funas-args-rule-list-def}$

lemma vars-trs-list-Nil [simp]:
 $\text{vars-trs-list } [] = []$ **unfolding** vars-trs-list-def **by** simp

context
fixes $R :: ('f, 'v) \text{trs}$
assumes $\text{wf-trs } R$
begin

lemma $\text{funas-term-subst-rhs}$:
assumes $\text{funas-trs } R \subseteq F$ **and** $(l, r) \in R$ **and** $\text{funas-term } (l \cdot \sigma) \subseteq F$
shows $\text{funas-term } (r \cdot \sigma) \subseteq F$
proof –
have $\text{vars-term } r \subseteq \text{vars-term } l$ **using** $\langle \text{wf-trs } R \rangle$ **and** $\langle (l, r) \in R \rangle$ **by** $(\text{auto simp: wf-trs-def})$
moreover have $\text{funas-term } l \subseteq F$ **and** $\text{funas-term } r \subseteq F$
using $\langle \text{funas-trs } R \subseteq F \rangle$ **and** $\langle (l, r) \in R \rangle$ **by** $(\text{auto simp: funas-defs})$ **force+**
ultimately show $?thesis$
using $\langle \text{funas-term } (l \cdot \sigma) \subseteq F \rangle$ **by** $(\text{force simp: funas-term-subst})$
qed

lemma vars-rule-lhs :
 $r \in R \implies \text{vars-rule } r = \text{vars-term } (\text{fst } r)$
using $\langle \text{wf-trs } R \rangle$ **by** $(\text{cases } r) (\text{auto simp: wf-trs-def vars-rule-def})$

end

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

lemma NF-trs-mono : $r \subseteq s \implies \text{NF-trs } s \subseteq \text{NF-trs } r$
by $(\text{rule NF-anti-mono}[\text{OF rstep-mono}])$

lemma NF-trs-union : $\text{NF-trs } (R \cup S) = \text{NF-trs } R \cap \text{NF-trs } S$
unfolding rstep-union **using** $\text{NF-anti-mono}[\text{of } - \text{rstep } R \cup \text{rstep } S]$ **by** auto

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

lemma $\text{NF-terms-anti-mono}$:
 $Q \subseteq Q' \implies \text{NF-terms } Q' \subseteq \text{NF-terms } Q$
by $(\text{rule NF-trs-mono, auto})$

```

lemma lhs-var-not-NF:
  assumes  $l \in T$  and is-Var  $l$  shows  $t \notin \text{NF-terms } T$ 
proof -
  from assms obtain  $x$  where  $l = \text{Var } x$  by (cases  $l$ , auto)
  let  $?\sigma = \text{subst } x \ t$ 
  from assms have  $l \notin \text{NF-terms } T$  by auto
  with NF-instance[of  $l \ ?\sigma \ \text{Id-on } T$ ]
    have  $l \cdot ?\sigma \notin \text{NF-terms } T$  by auto
  then show ?thesis by (simp add: l subst-def)
qed

lemma not-NF-termsE[elim]:
  assumes  $s \notin \text{NF-terms } Q$ 
  obtains  $l \ C \ \sigma$  where  $l \in Q$  and  $s = C\langle l \cdot \sigma \rangle$ 
proof -
  from assms obtain  $t$  where  $(s, t) \in \text{rstep } (\text{Id-on } Q)$  by auto
  with  $\langle \bigwedge l \ C \ \sigma. [l \in Q; s = C\langle l \cdot \sigma \rangle] \implies \text{thesis} \rangle$  show ?thesis by auto
qed

lemma notin-NF-E [elim]:
  fixes  $R :: ('f, 'v) \text{trs}$ 
  assumes  $t \notin \text{NF-trs } R$ 
  obtains  $C \ l$  and  $\sigma :: ('f, 'v) \text{subst}$  where  $l \in \text{lhss } R$  and  $t = C\langle l \cdot \sigma \rangle$ 
proof -
  assume  $1: \bigwedge l \ C \ (\sigma :: ('f, 'v) \text{subst}). l \in \text{lhss } R \implies t = C\langle l \cdot \sigma \rangle \implies \text{thesis}$ 
  from assms obtain  $u$  where  $(t, u) \in \text{rstep } R$  by (auto simp: NF-def)
  then obtain  $C \ \sigma \ l \ r$  where  $(l, r) \in R$  and  $t = C\langle l \cdot \sigma \rangle$  by blast
  with  $1$  show ?thesis by force
qed

lemma NF-ctxt-subst:  $\text{NF-terms } Q = \{t. \neg (\exists \ C \ q \ \sigma. t = C\langle q \cdot \sigma \rangle \wedge q \in Q)\}$  (is
   $- = ?R$ )
proof -
  {
    fix  $t$ 
    assume  $t \notin ?R$ 
    then obtain  $C \ q \ \sigma$  where  $t = C\langle q \cdot \sigma \rangle$  and  $q: q \in Q$  by auto
    have  $(t, t) \in \text{rstep } (\text{Id-on } Q)$ 
      unfolding  $t$  using  $q$  by auto
    then have  $t \notin \text{NF-terms } Q$  by auto
  }
  moreover
  {
    fix  $t$ 
    assume  $t \notin \text{NF-terms } Q$ 
    then obtain  $C \ q \ \sigma$  where  $t = C\langle q \cdot \sigma \rangle$  and  $q: q \in Q$  by auto
    then have  $t \notin ?R$  by auto
  }
  ultimately show ?thesis by auto

```

qed

lemma *some-NF-imp-no-Var*:

assumes $t \in \text{NF-terms } Q$

shows $\text{Var } x \notin Q$

proof

assume $\text{Var } x \in Q$

with *assms*[*unfolded NF-ctxt-subst*] **have** $\bigwedge \sigma \ C. \ t \neq C \langle \sigma \ x \rangle$ **by** *force*

from *this*[*of Hole* $\lambda \ -. \ t$] **show** *False* **by** *simp*

qed

lemma *NF-map-vars-term-inj*:

assumes *inj*: $\bigwedge x. n \ (m \ x) = x$ **and** *NF*: $t \in \text{NF-terms } Q$

shows $(\text{map-vars-term } m \ t) \in \text{NF-terms } (\text{map-vars-term } m \ ` \ Q)$

proof (*rule ccontr*)

assume $\neg \text{?thesis}$

then obtain *u* **where** $(\text{map-vars-term } m \ t, u) \in \text{rstep } (\text{Id-on } (\text{map-vars-term } m \ ` \ Q))$ **by** *blast*

then obtain *ml mr C* σ **where** *in-mR*: $(ml, mr) \in \text{Id-on } (\text{map-vars-term } m \ ` \ Q)$

and *mt*: $\text{map-vars-term } m \ t = C \langle ml \cdot \sigma \rangle$ **by** *best*

let *?m* = *n*

from *in-mR* **obtain** *l r* **where** $(l, r) \in \text{Id-on } Q$ **and** *ml*: $ml = \text{map-vars-term } m \ l$ **by** *auto*

have $t = \text{map-vars-term } ?m \ (\text{map-vars-term } m \ t)$ **by** (*simp add: map-vars-term-inj-compose*[*of n m, OF inj*])

also have $\dots = \text{map-vars-term } ?m \ (C \langle ml \cdot \sigma \rangle)$ **by** (*simp add: mt*)

also have $\dots = (\text{map-vars-ctxt } ?m \ C) \langle \text{map-vars-term } ?m \ (\text{map-vars-term } m \ l \cdot \sigma) \rangle$

by (*simp add: map-vars-term-ctxt-commute ml*)

also have $\dots = (\text{map-vars-ctxt } ?m \ C) \langle l \cdot (\text{map-vars-subst-ran } ?m \ (\sigma \circ m)) \rangle$

by (*simp add: apply-subst-map-vars-term map-vars-subst-ran*)

finally show *False* **using** *NF* **and** $\langle l, r \rangle \in \text{Id-on } Q$ **by** *auto*

qed

lemma *notin-NF-terms*: $t \in Q \implies t \notin \text{NF-terms } Q$

using *lhs-notin-NF-rstep*[*of t t Id-on Q*] **by** (*simp add: Id-on-iff*)

lemma *NF-termsI* [*intro*]:

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

shows $t \in \text{NF-terms } Q$

by (*rule ccontr, rule not-NF-termsE* [*OF - NF*])

lemma *NF-args-imp-NF*:

assumes *ss*: $\bigwedge s. s \in \text{set } ss \implies s \in \text{NF-terms } Q$

and *someNF*: $t \in \text{NF-terms } Q$

and *root*: $\text{Some } (f, \text{length } ss) \notin \text{root } ` \ Q$

shows $(\text{Fun } f \ ss) \in \text{NF-terms } Q$

proof

```

fix C l σ
assume id: Fun f ss = C ⟨ l · σ ⟩ and l: l ∈ Q
show False
proof (cases C)
  case Hole
  with id have id: Fun f ss = l · σ by simp
  show False
  proof (cases l)
    case (Fun g ls)
    with id have fg: f = g and ss: ss = map (λ s. s · σ) ls by auto
    from arg-cong[OF ss, of length] have len: length ss = length ls by simp
    from l[unfolded Fun] root[unfolded fg len] show False by force
  next
    case (Var x)
    from some-NF-imp-no-Var[OF someNF] Var l show False by auto
  qed
next
case (More g bef D aft)
note id = id[unfolded More]
from id have NF: ss ! length bef = D ⟨ l · σ ⟩ by auto
from id have mem: ss ! length bef ∈ set ss by auto
from ss[OF mem, unfolded NF-ctxt-subst NF] l show False by auto
qed
qed

lemma NF-Var-is-Fun:
  assumes Q: Ball Q is-Fun
  shows Var x ∈ NF-terms Q
proof
  fix C l σ
  assume x: Var x = C ⟨ l · σ ⟩ and l: l ∈ Q
  from l Q obtain f ls where l: l = Fun f ls by (cases l, auto)
  then show False using x by (cases C, auto)
qed

lemma NF-terms-lhss [simp]: NF-terms (lhss R) = NF (rstep R)
proof
  show NF (rstep R) ⊆ NF-terms (lhss R) by force
next
  show NF-terms (lhss R) ⊆ NF (rstep R)
  proof
    fix s assume NF: s ∈ NF-terms (lhss R)
    show s ∈ NF (rstep R)
    proof (rule ccontr)
      assume s ∉ NF (rstep R)
      then obtain t where (s, t) ∈ rstep R by auto
      then obtain l r C σ where (l, r) ∈ R and s: s = C⟨l · σ⟩ by auto
      then have (l, l) ∈ Id-on (lhss R) by force
      then have (s, s) ∈ rstep (Id-on (lhss R)) unfolding s by auto
    qed
  qed

```

with *NF* show *False* by *auto*
 qed
 qed
 qed

fun *fun-poss-list* :: ('f, 'v) term \Rightarrow pos list
where

fun-poss-list (Var *x*) = [] |
fun-poss-list (Fun *f* *ss*) = ([] # concat (map (λ (*i*, *ps*).
 map ((#) *i*) *ps*) (zip [0..ss] (map *fun-poss-list* *ss*))))

lemma *set-fun-poss-list* [*simp*]:

set (*fun-poss-list* *t*) = *fun-poss* *t*

by (*induct* *t*; *auto simp: UNION-set-zip*)

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

lemma *args-SN-on-relstep-nrrstep-imp-args-SN-on*:

assumes *SN*: $\bigwedge u. s \triangleright u \Longrightarrow \text{SN-on } (\text{relstep } R \ E) \ \{u\}$

and *st*: (*s*, *t*) \in *nrrstep* (*R* \cup *E*)

and *supt*: *t* \triangleright *u*

shows *SN-on* (*relstep* *R* *E*) {*u*}

proof –

from *nrrstepE*[*OF st*] **obtain** *C l r σ* **where** *C* $\neq \square$ **and** *lr*: (*l*, *r*) \in *R* \cup *E*

and *s*: *s* = *C*⟨*l* \cdot σ ⟩ **and** *t*: *t* = *C*⟨*r* \cdot σ ⟩ **by** *blast*

then obtain *f bef C aft* **where** *s*: *s* = *Fun* *f* (*bef* @ *C*⟨*l* \cdot σ ⟩ # *aft*) **and** *t*: *t* =
Fun *f* (*bef* @ *C*⟨*r* \cdot σ ⟩ # *aft*)

by (*cases C*, *auto*)

let *?ts* = *bef* @ *C*⟨*r* \cdot σ ⟩ # *aft*

let *?ss* = *bef* @ *C*⟨*l* \cdot σ ⟩ # *aft*

from *supt* **obtain** *D* **where** *t* = *D*⟨*u*⟩ **and** *D* $\neq \square$ **by** *auto*

then obtain *bef' aft' D* **where** *t'*: *t* = *Fun* *f* (*bef'* @ *D*⟨*u*⟩ # *aft'*) **unfolding** *t*
by (*cases D*, *auto*)

have *D*⟨*u*⟩ \triangleright *u* **by** *auto*

then have *supt*: $\bigwedge s. s \triangleright D\langle u \rangle \Longrightarrow s \triangleright u$ **by** (*metis supt-supteq-trans*)

show *SN-on* (*relstep* *R* *E*) {*u*}

proof (*cases D*⟨*u*⟩ \in *set* *?ss*)

case *True*

then have *s* \triangleright *D*⟨*u*⟩ **unfolding** *s* **by** *auto*

then have *s* \triangleright *u* **by** (*rule supt*)

with *SN* **show** *?thesis* **by** *auto*

next

case *False*

have *D*⟨*u*⟩ \in *set* *?ts* **using** *arg-cong*[*OF t'*[*unfolded t*], *of args*] **by** *auto*

with *False* **have** *Du*: *D*⟨*u*⟩ = *C*⟨*r* \cdot σ ⟩ **by** *auto*

have *s* \triangleright *C*⟨*l* \cdot σ ⟩ **unfolding** *s* **by** *auto*

with *SN* **have** *SN-on* (*relstep* *R* *E*) {*C*⟨*l* \cdot σ ⟩} **by** *auto*

from *step-preserves-SN-on-relto*[*OF - this*, *of C*⟨*r* \cdot σ ⟩] *lr*

have *SN*: *SN-on* (*relstep* *R* *E*) {*D*⟨*u*⟩} **using** *Du* **by** *auto*

show *?thesis*
by (rule *ctxt-closed-SN-on-subt*[*OF ctxt.closed-relto SN*], *auto*)
qed
qed

lemma *Tinf-nrrstep*:
assumes *tinf*: $s \in Tinf \text{ (relstep } R \ E)$ **and** *st*: $(s, t) \in nrrstep \ (R \cup E)$
and *t*: $\neg SN\text{-on} \text{ (relstep } R \ E) \ \{t\}$
shows $t \in Tinf \text{ (relstep } R \ E)$
unfolding *Tinf-def*
by (intro *CollectI conjI*[*OF t*] *allI impI*)
(rule *args-SN-on-relstep-nrrstep-imp-args-SN-on*[*OF - st*],
insert *tinf*[*unfolded Tinf-def*], *auto*)

lemma *subterm-preserves-SN-on-relstep*:
 $SN\text{-on} \text{ (relstep } R \ E) \ \{s\} \implies s \triangleright t \implies SN\text{-on} \text{ (relstep } R \ E) \ \{t\}$
using *SN-imp-SN-subt* [*of rstep (rstep ((rstep E)*) O rstep R O rstep ((rstep E)*)*)]
by (*simp only: rstep-relcomp-idemp2*) (*simp only: rstep-rtrancl-idemp*)

inductive-set *rstep-rule* :: $(f, v) \text{ rule} \implies (f, v) \text{ term rel for } \varrho$
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*]:
 $s = C\langle l \cdot \sigma \rangle \implies t = C\langle r \cdot \sigma \rangle \implies (s, t) \in \text{rstep-rule} \ (l, r)$
by (*auto simp: rstep-rule.simps*)

lemma *rstep-rule-ctxt*:
 $(s, t) \in \text{rstep-rule } \varrho \implies (C\langle s \rangle, C\langle t \rangle) \in \text{rstep-rule } \varrho$
using *rstep-rule.rule* [*of C⟨s⟩ C ◦_c D ϱ - C⟨t⟩ for D*]
by (*auto elim: rstep-rule.cases simp: ctxt-of-pos-term-append*)

lemma *rstep-rule-subst*:
assumes $(s, t) \in \text{rstep-rule } \varrho$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{rstep-rule } \varrho$
using *assms*
proof (*cases*)
case (*rule C τ*)
then show *?thesis*
using *rstep-rule.rule* [*of s · σ - ϱ τ ◦_s σ*]
by (*auto elim!: rstep-rule.cases simp: ctxt-of-pos-term-subst*)
qed

lemma *rstep-rule-imp-rstep*:
 $\varrho \in R \implies (s, t) \in \text{rstep-rule } \varrho \implies (s, t) \in \text{rstep } R$
by (*force elim: rstep-rule.cases*)

lemma *rstep-imp-rstep-rule*:

assumes $(s, t) \in rstep\ R$
obtains $l\ r$ **where** $(l, r) \in R$ **and** $(s, t) \in rstep\text{-rule}\ (l, r)$
using *assms* **by** *blast*

lemma *term-subst-rstep*:

assumes $\bigwedge x. x \in vars\text{-term}\ t \implies (\sigma\ x, \tau\ x) \in rstep\ R$
shows $(t \cdot \sigma, t \cdot \tau) \in (rstep\ R)^*$
using *assms*
proof (*induct t*)
case (*Fun f ts*)
{ **fix** t_i
assume $t_i: t_i \in set\ ts$
with *Fun(2)* **have** $\bigwedge x. x \in vars\text{-term}\ t_i \implies (\sigma\ x, \tau\ x) \in rstep\ R$ **by** *auto*
from *Fun(1)* [*OF t_i this*] **have** $(t_i \cdot \sigma, t_i \cdot \tau) \in (rstep\ R)^*$ **by** *blast*
}
then show *?case* **by** (*simp add: args-rsteps-imp-rsteps*)
qed (*auto*)

lemma *term-subst-rsteps*:

assumes $\bigwedge x. x \in vars\text{-term}\ t \implies (\sigma\ x, \tau\ x) \in (rstep\ R)^*$
shows $(t \cdot \sigma, t \cdot \tau) \in (rstep\ R)^*$
by (*metis assms rstep-rtrancl-idemp rtrancl-idemp term-subst-rstep*)

lemma *term-subst-rsteps-join*:

assumes $\bigwedge y. y \in vars\text{-term}\ u \implies (\sigma_1\ y, \sigma_2\ y) \in (rstep\ R)^\downarrow$
shows $(u \cdot \sigma_1, u \cdot \sigma_2) \in (rstep\ R)^\downarrow$
using *assms*
proof –
{ **fix** x
assume $x \in vars\text{-term}\ u$
from *assms* [*OF this*] **have** $\exists \sigma. (\sigma_1\ x, \sigma\ x) \in (rstep\ R)^* \wedge (\sigma_2\ x, \sigma\ x) \in (rstep\ R)^*$ **by** *auto*
}
then have $\forall x \in vars\text{-term}\ u. \exists \sigma. (\sigma_1\ x, \sigma\ x) \in (rstep\ R)^* \wedge (\sigma_2\ x, \sigma\ x) \in (rstep\ R)^*$ **by** *blast*
then obtain s **where** $\forall x \in vars\text{-term}\ u. (\sigma_1\ x, (s\ x)\ x) \in (rstep\ R)^* \wedge (\sigma_2\ x, (s\ x)\ x) \in (rstep\ R)^*$ **by** *metis*
then obtain σ **where** $\forall x \in vars\text{-term}\ u. (\sigma_1\ x, \sigma\ x) \in (rstep\ R)^* \wedge (\sigma_2\ x, \sigma\ x) \in (rstep\ R)^*$ **by** *fast*
then have $(u \cdot \sigma_1, u \cdot \sigma) \in (rstep\ R)^* \wedge (u \cdot \sigma_2, u \cdot \sigma) \in (rstep\ R)^*$ **using** *term-subst-rsteps* **by** *metis*
then show *?thesis* **by** *blast*
qed

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

by (*auto simp: funas-defs*)

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

$\{(r, l)\} \ p)$
proof –
 from *assms* obtain σ where $\text{step}:t = (\text{ctxt-of-pos-term } p \ s) \langle r \cdot \sigma \rangle \ p \in \text{poss } s \ s$
 $\mid - p = l \cdot \sigma$
 unfolding *rstep-pos.simps* by *auto*
 with *replace-at-below-poss*[*of p s p*] have $pt:p \in \text{poss } t$ by *auto*
 with *step ctxt-supt-id*[*OF step(2)*] have $s = (\text{ctxt-of-pos-term } p \ t) \langle l \cdot \sigma \rangle$
 by (*simp add: ctxt-of-pos-term-replace-at-below*)
 with *step ctxt-supt-id*[*OF pt*] show ?thesis unfolding *rstep-pos.simps*
 by (*metis pt replace-at-subt-at singletonI*)
qed

lemma *conversion-ctxt-closed*: $(s, t) \in (\text{rstep } R)^{\leftrightarrow*} \implies (C\langle s \rangle, C\langle t \rangle) \in (\text{rstep } R)^{\leftrightarrow*}$
 using *rsteps-closed-ctxt* unfolding *conversion-def*
 by (*simp only: rstep-simps(5)[symmetric]*)

lemma *conversion-subst-closed*:
 $(s, t) \in (\text{rstep } R)^{\leftrightarrow*} \implies (s \cdot \sigma, t \cdot \sigma) \in (\text{rstep } R)^{\leftrightarrow*}$
 using *rsteps-closed-subst* unfolding *conversion-def*
 by (*simp only: rstep-simps(5)[symmetric]*)

lemma *rstep-simulate-conv*:
 assumes $\bigwedge l \ r. (l, r) \in S \implies (l, r) \in (\text{rstep } R)^{\leftrightarrow*}$
 shows $(\text{rstep } S) \subseteq (\text{rstep } R)^{\leftrightarrow*}$
proof
 fix $s \ t$
 assume $(s, t) \in \text{rstep } S$
 then obtain $l \ r \ C \ \sigma$ where $s: s = C\langle l \cdot \sigma \rangle$ and $t:t = C\langle r \cdot \sigma \rangle$ and $lr: (l, r) \in S$
 unfolding *rstep-iff-rstep-r-c-s* *rstep-r-c-s-def* by *auto*
 with *assms* have $(l, r) \in (\text{rstep } R)^{\leftrightarrow*}$ by *auto*
 then show $(s, t) \in (\text{rstep } R)^{\leftrightarrow*}$ using *conversion-ctxt-closed* *conversion-subst-closed*
 $s \ t$ by *metis*
qed

lemma *symcl-simulate-conv*:
 assumes $\bigwedge l \ r. (l, r) \in S \implies (l, r) \in (\text{rstep } R)^{\leftrightarrow*}$
 shows $(\text{rstep } S)^{\leftrightarrow} \subseteq (\text{rstep } R)^{\leftrightarrow*}$
 using *rstep-simulate-conv*[*OF assms*]
 by *auto* (*metis conversion-inv subset-iff*)

lemma *conv-union-simulate*:
 assumes $\bigwedge l \ r. (l, r) \in S \implies (l, r) \in (\text{rstep } R)^{\leftrightarrow*}$
 shows $(\text{rstep } (R \cup S))^{\leftrightarrow*} = (\text{rstep } R)^{\leftrightarrow*}$
proof
 show $(\text{rstep } (R \cup S))^{\leftrightarrow*} \subseteq (\text{rstep } R)^{\leftrightarrow*}$
 unfolding *conversion-def*

```

proof
  fix  $s\ t$ 
  assume  $(s, t) \in ((rstep\ (R \cup S))^{\leftrightarrow})^*$ 
  then show  $(s, t) \in ((rstep\ R)^{\leftrightarrow})^*$ 
  proof (induct rule: rtrancl-induct)
    case ( $step\ u\ t$ )
    then have  $(u, t) \in (rstep\ R)^{\leftrightarrow} \vee (u, t) \in (rstep\ S)^{\leftrightarrow}$  by auto
    then show ?case
    proof
      assume  $(u, t) \in (rstep\ R)^{\leftrightarrow}$ 
      with step show ?thesis using rtrancl-into-rtrancl by metis
    next
      assume  $(u, t) \in (rstep\ S)^{\leftrightarrow}$ 
      with symcl-simulate-conv[OF assms] have  $(u, t) \in (rstep\ R)^{\leftrightarrow*}$  by auto
      with step show ?thesis by auto
    qed
  qed simp
qed
next
  show  $(rstep\ R)^{\leftrightarrow*} \subseteq (rstep\ (R \cup S))^{\leftrightarrow*}$ 
  unfolding conversion-def
  using rstep-union rtrancl-mono sup.cobounded1 symcl-Un
  by metis
qed

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

end

```

4 Overloading for the Sharp Symbol

```

theory Sharp-Syntax
imports
  Main
  HOL-Library.Adhoc-Overloading
  First-Order-Rewriting.Trs
begin

consts SHARP :: ' $a \Rightarrow 'b$  ( $\sharp$ )

locale sharp-syntax =
  fixes shp :: ' $f \Rightarrow 'f$ 
begin

adhoc-overloading SHARP shp

end

context

```

```

fixes shp :: 'f ⇒ 'f
begin

interpretation sharp-syntax .

fun sharp-term :: ('f, 'v) term ⇒ ('f, 'v) term
where
  sharp-term (Var x) = Var x |
  sharp-term (Fun f ss) = Fun (# f) ss

fun sharp-ctxt :: ('f, 'v) ctxt ⇒ ('f, 'v) ctxt
where
  sharp-ctxt □ = □ |
  sharp-ctxt (More f ss1 C ss2) = More (# f) ss1 C ss2

abbreviation sharp-sig :: ('f × nat) set ⇒ ('f × nat) set
where
  sharp-sig ≡ image (λ(f, n). (# f, n))
end

context sharp-syntax
begin

adhoc-overloading
  SHARP sharp-term shp sharp-ctxt shp sharp-sig shp

end

context
  fixes shp :: 'f ⇒ 'f
begin

interpretation sharp-syntax .

lemma sharp-term-ctxt-apply [simp]:
  C ≠ □ ⇒ #(C⟨t⟩) = (# C)⟨t⟩
  by (cases C) simp-all

lemma supt-sharp-term-subst [simp]:
  # s · σ ▷ t ⟷ s · σ ▷ t
  by (cases s) auto

end

lemma sharp-term-id [simp]:
  sharp-term id t = t
  sharp-term (λx. x) t = t
  by (induct t) simp-all

```

A theory on first-order term rewrite systems (TRSs).

```

context
  fixes shp :: 'f  $\Rightarrow$  'f
begin

interpretation sharp-syntax .

abbreviation sharp-trs :: ('f, 'v) trs  $\Rightarrow$  ('f, 'v) trs
where
  sharp-trs R  $\equiv$  dir-image R  $\#$ 

end

context sharp-syntax
begin

adhoc-overloading
  SHARP sharp-trs shp

end

context
  fixes shp :: 'f  $\Rightarrow$  'f
begin

interpretation sharp-syntax .

definition DP-on :: 'f sig  $\Rightarrow$  ('f, 'v) trs  $\Rightarrow$  ('f, 'v) trs
where
  DP-on F R = {(s, t).  $\exists$  l r h us. s =  $\#$  l  $\wedge$  t =  $\#$  (Fun h us)  $\wedge$ 
    (l, r)  $\in$  R  $\wedge$  r  $\supseteq$  Fun h us  $\wedge$  (h, length us)  $\in$  F  $\wedge$   $\neg$  l  $\triangleright$  Fun h us}

abbreviation DP R  $\equiv$  DP-on {f. defined R f} R

lemma nrrstep-imp-sharp-nrrstep: assumes (s, t)  $\in$  nrrstep R
shows ( $\#$  s,  $\#$  t)  $\in$  nrrstep R
proof –
  from assms obtain C l r  $\sigma$  where C  $\neq$   $\square$  and (l, r)  $\in$  R
    and *: s = C  $\langle$  l  $\cdot$   $\sigma$   $\rangle$  t = C  $\langle$  r  $\cdot$   $\sigma$   $\rangle$ 
    by (auto elim: nrrstepE)
  then obtain D f ss ts where C = More f ss D ts
    and s = Fun f (ss @ D  $\langle$  l  $\cdot$   $\sigma$   $\rangle$   $\#$  ts) by (cases C) (auto elim: sharp-term.elims)
  moreover with  $\langle$  t = C  $\langle$  r  $\cdot$   $\sigma$   $\rangle$   $\rangle$  have t = Fun f (ss @ D  $\langle$  r  $\cdot$   $\sigma$   $\rangle$   $\#$  ts)
    using assms by auto
  moreover define C' where C' = More ( $\#$  f) ss D ts
  ultimately have  $\#$  s = C'  $\langle$  l  $\cdot$   $\sigma$   $\rangle$  and  $\#$  t = C'  $\langle$  r  $\cdot$   $\sigma$   $\rangle$  by simp+
  moreover have C'  $\neq$   $\square$  using  $\langle$  C  $\neq$   $\square$   $\rangle$  by (simp add: C'-def)
  ultimately show ( $\#$  s,  $\#$  t)  $\in$  nrrstep R using  $\langle$  (l, r)  $\in$  R  $\rangle$  by (auto simp: nrrstep-def')
qed

```

```

lemma nrrstep-imp-sharp-rstep:
  assumes  $(s, t) \in \text{nrrstep } R$ 
  shows  $(\# s, \# t) \in \text{rstep } R$ 
  using nrrstep-imp-sharp-nrrstep[OF assms] by (rule nrrstep-imp-rstep)

lemma nrrsteps-imp-sharp-rsteps:
   $(s, t) \in (\text{nrrstep } R)^* \implies (\# s, \# t) \in (\text{rstep } R)^*$ 
proof (induct rule: rtrancl-induct)
  case (step a b)
  from  $\langle a, b \rangle \in \text{nrrstep } R$  have  $(\# a, \# b) \in \text{rstep } R$ 
    by (rule nrrstep-imp-sharp-rstep)
  with step show ?case by auto
qed simp

lemma finiteR-imp-finiteDP:
  assumes finite R
  shows finite (DP-on F R)
proof –
  have fS: finite  $\{(l, r, u). \exists h \text{ us. } u = \text{Fun } h \text{ us} \wedge (l, r) \in R \wedge r \triangleright u \wedge (h, \text{length } \text{us}) \in F \wedge \neg(l \triangleright u)\}$  (is
finite ?S)
  using assms by (rule finite-imp-finite-DP-on')
  let ?f =  $\lambda(x :: ('f, 'v) \text{ term}, y, z :: ('f, 'v) \text{ term}). (\# x, \# z)$ 
  have eq1:  $(\bigcup y \in ?S. \{x. x = ?f y\}) = ?f ' ?S$  by blast
  with fS have finite(?f ' ?S) by auto
  have DP-on F R = ?f ' ?S (is ?DP = ?T)
  proof
    show ?DP  $\subseteq$  ?T
    proof
      fix x assume  $x \in ?DP$ 
      then obtain l r h us
        where fst x =  $\# l$  and snd x =  $\# (\text{Fun } h \text{ us})$ 
        and  $(l, r) \in R$   $r \triangleright \text{Fun } h \text{ us}$  and  $(h, \text{length } \text{us}) \in F$  and  $\neg(l \triangleright \text{Fun } h \text{ us})$ 
        by (auto simp: DP-on-def split-def)
      then have  $(l, r, \text{Fun } h \text{ us}) \in ?S$  by auto
      then show  $x \in ?T$  unfolding eq1[symmetric]
      proof (rule UN-I)
        have  $x = (\text{fst } x, \text{snd } x)$  by simp
        then have  $x = (\# l, \# (\text{Fun } h \text{ us}))$ 
        unfolding  $\langle \text{fst } x = \# l \rangle \langle \text{snd } x = \# (\text{Fun } h \text{ us}) \rangle$  .
        then show  $x \in \{x. x = ?f (l, r, \text{Fun } h \text{ us})\}$  by auto
      qed
    qed
  next
    show ?T  $\subseteq$  ?DP
    proof
      fix x assume  $x \in ?T$ 
      then have  $x \in (\bigcup y \in ?S. \{x. x = ?f y\})$  unfolding eq1 .
    qed

```

then obtain y where $y \in ?S$ and $x = ?f y$ by *fast*
 then obtain $l r h us$ where $y = (l, r, Fun h us)$ and $dp: (l, r) \in R$
 and $r \supseteq Fun h us$ and $(h, length us) \in F$ and $\neg (l \triangleright Fun h us)$ by *blast*
 moreover with $\langle x = ?f y \rangle$ have $x = (\# l, \# (Fun h us))$ by *auto*
 ultimately show $x \in ?DP$ using *dp unfolding DP-on-def* by *auto*
 qed
 qed
 with *fS* show *?thesis* by *auto*
 qed

lemma *vars-sharp-eq-vars [simp]*: *vars-term* $(\# t) = vars-term t$
 by (*induct t*) *auto*

lemma *wf-trs-imp-wf-DP-on*:
 assumes *wf-trs* R
 shows *wf-trs* (*DP-on* $F R$)
 unfolding *wf-trs-def*
 proof (*intro allI impI*)
 fix $s t$
 assume $(s, t) \in DP-on F R$
 then obtain $l r h us$ where $s = \# l$ and $t = \# (Fun h us)$ and $(l, r) \in R$
 and $r \supseteq (Fun h us) \neg (l \triangleright Fun h us)$
 by (*auto simp: DP-on-def*)
 from $\langle wf-trs R \rangle$ and $\langle (l, r) \in R \rangle$
 have $\exists f ss. l = Fun f ss$ and *vars-term* $r \subseteq vars-term l$ by (*auto simp:*
wf-trs-def)
 from $\langle \exists f ss. l = Fun f ss \rangle$ obtain $f ss$ where $l = Fun f ss$ by *auto*
 then have $s = Fun (\# f) ss$ unfolding $\langle s = \# l \rangle$ by *simp*
 then have $\exists f ss. s = Fun f ss$ by *auto*
 from $\langle r \supseteq Fun h us \rangle$ have *vars-term* $(Fun h us) \leq vars-term r$ by (*induct rule:*
supteq.induct) *auto*
 then have *vars-term* $t \subseteq vars-term s$ unfolding $\langle s = \# l \rangle$
 and $\langle t = \# (Fun h us) \rangle$ *vars-sharp-eq-vars* using $\langle vars-term r \leq vars-term l \rangle$
 by *simp*
 from $\langle \exists f ss. s = Fun f ss \rangle \langle vars-term t \subseteq vars-term s \rangle$
 show $(\exists f ss. s = Fun f ss) \wedge vars-term t \subseteq vars-term s$ by *simp*
 qed

lemma *sharp-eq-imp-eq*:
 fixes $s :: ('f, 'v) term$
 assumes *inj* $(\# :: 'f \Rightarrow 'v)$
 shows $\# s = \# t \implies s = t$
 proof (*cases s*)
 case (*Var x*)
 assume $\# s = \# t$ with *Var* show *?thesis* by (*induct t*) *auto*
 next
 case (*Fun f ss*)
 assume $\# s = \# t$
 with *Fun* have $\# (Fun f ss) = \# t$ by *simp*

then obtain $g\ ts$ where $t: t = \text{Fun } g\ ts$ by (induct t) auto
 with $\langle \# s = \# t \rangle$ have $\text{Fun } (\# f)\ ss = \text{Fun } (\# g)\ ts$ unfolding $\langle s = \text{Fun } f\ ss \rangle \langle t = \text{Fun } g\ ts \rangle$
 by simp
 then have $f = g$ and $ss = ts$ using $\langle \text{inj } (\# :: 'f \Rightarrow 'f) \rangle [\text{unfolded inj-on-def}]$ by auto
 then show ?thesis unfolding Fun t by simp
 qed

lemma *DP-on-step-in-R*:

fixes $R :: ('f, 'v)\ trs$ and $v :: ('f, 'v)\ term \Rightarrow 'v$
 assumes $(s, t) \in \text{DP-on } F\ R$ and inj: $\text{inj } (\# :: 'f \Rightarrow 'f)$
 shows $\exists C. \text{funas-ctxt } C \subseteq \text{funas-trs } R \wedge$
 $(\text{sharp-term } (\text{the-inv } \#)\ s, C(\text{sharp-term } (\text{the-inv } \#)\ t)) \in R$
proof –
 let $?us = \text{sharp-term } (\text{the-inv } (\# :: 'f \Rightarrow 'f))$
 from *assms* obtain $l\ r\ f\ ts$
 where $s: s = \# l$ and $t: t = \# (\text{Fun } f\ ts)$
 and $R: (l, r) \in R$ and $\text{sub}: r \supseteq \text{Fun } f\ ts$ unfolding *DP-on-def* *supt-supteq-conv*
 by auto
 from *sub* obtain C where $r: r = C(\text{Fun } f\ ts)$ by auto
 from *rhs-wf* [*OF* R *subset-refl*] have $\text{funas-term } r \subseteq \text{funas-trs } R$.
 then have $\text{funas-term } (C(\text{Fun } f\ ts)) \subseteq \text{funas-trs } R$ unfolding r .
 then have $\text{funas-ctxt } C \subseteq \text{funas-trs } R$ and $\text{funas-term } (\text{Fun } f\ ts) \subseteq \text{funas-trs } R$ by auto
 from *lhs-wf* [*OF* R *subset-refl*] have $\text{funas-term } l \subseteq \text{funas-trs } R$.
 have $us: ?us\ s = l$ unfolding s by (cases l , auto simp: *the-inv-f-f* [*OF* *inj*])
 have $ut: ?us\ t = \text{Fun } f\ ts$ unfolding t by (simp add: *the-inv-f-f* [*OF* *inj*])
 from R have $(?us\ s, C(?us\ t)) \in R$ unfolding $us\ ut\ r$.
 with $\langle \text{funas-ctxt } C \subseteq \text{funas-trs } R \rangle$ show ?thesis by best
 qed

lemma *sharp-rrstep-imp-rstep*:

assumes *rrstep*: $(\# s, \# t) \in \text{subst.closure } (\text{DP-on } F\ R)$ and *inj* $(\# :: 'f \Rightarrow 'f)$
 and *wf-trs* R
 shows $\exists C. (s, C\langle t \rangle) \in \text{rstep } R$
proof –
 from $\langle \text{wf-trs } R \rangle$ have *wf-trs* $(\text{DP-on } F\ R)$ by (rule *wf-trs-imp-wf-DP-on*)
 from *rrstep* obtain $l\ r\ \sigma$ where $(l, r) \in \text{DP-on } F\ R$ and $ss: \# s = l \cdot \sigma$ and $st: \# t = r \cdot \sigma$
 by (induct, auto)
 from $\langle (l, r) \in \text{DP-on } F\ R \rangle$ obtain $l'\ r'\ h'\ us'$
 where $l: l = \# l'$ and $r: r = \# (\text{Fun } h'\ us')$ (is $r = \# ?u'$)
 and $l'r': (l', r') \in R$ and $r' \supseteq (\text{Fun } h'\ us')$ and $\neg l' \triangleright \text{Fun } h'\ us'$
 unfolding *DP-on-def* by auto
 from $\langle \text{wf-trs } R \rangle$ and $\langle (l', r') \in R \rangle$ obtain $f'\ ss'$ where $l': l' = \text{Fun } f'\ ss'$ using *wf-trs-imp-lhs-Fun* by best
 from $\langle r' \supseteq ?u' \rangle$ obtain C where $r': r' = C(?u')$ by best
 have $ss: \# s = (\text{Fun } (\# f')\ ss') \cdot \sigma$ unfolding $ss\ l\ l'$ by simp

then obtain f where $s: s = (\text{Fun } f \text{ } ss') \cdot \sigma$ by $(\text{cases } s, \text{auto})$
from $s \text{ } ss$ have $\# f = \# f'$ by simp
with $\langle \text{inj } (\# :: 'f \Rightarrow 'f) \rangle [\text{unfolded inj-on-def}]$ have $f: f = f'$ by simp
have $ts: \# t = (\text{Fun } (\# h') \text{ } us') \cdot \sigma$ unfolding $st \text{ } r$ by simp
then obtain h where $t: t = (\text{Fun } h \text{ } us') \cdot \sigma$ by $(\text{cases } t, \text{auto})$
from $t \text{ } ts$ have $\# h = \# h'$ by simp
with $\langle \text{inj } (\# :: 'f \Rightarrow 'f) \rangle [\text{unfolded inj-on-def}]$ have $h: h = h'$ by simp
show $?thesis$
by $(\text{rule exI}[\text{of} - C \cdot_c \sigma], \text{unfold } s \text{ } t \text{ } f \text{ } h, \text{rule rstepI}[\text{OF } l'r', \text{of} - \square \sigma], \text{unfold } l'$
 $r', \text{simp}, \text{simp})$
qed

definition $DP\text{-simple} :: 'f \text{ sig} \Rightarrow ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ trs}$
where

$DP\text{-simple } D \text{ } R = \{(s, t).$
 $\exists l \text{ } r \text{ } h \text{ } us. s = \# l \wedge t = \# (\text{Fun } h \text{ } us) \wedge (l, r) \in R \wedge (h, \text{length } us) \in D \wedge r \supseteq$
 $\text{Fun } h \text{ } us\}$

lemma $DP\text{-on-subset-}DP\text{-simple}: DP\text{-on } F \text{ } R \subseteq DP\text{-simple } F \text{ } R$
by $(\text{auto simp: } DP\text{-on-def } DP\text{-simple-def})$

lemma $\text{funas-}DP\text{-simple-subset}:$

$\text{funas-trs } (DP\text{-simple } D \text{ } R) \subseteq \text{funas-trs } R \cup \# (\text{funas-trs } R)$
(is $?F \subseteq ?H \cup ?I$)

proof (rule subrelI)

fix $f \text{ } n$
assume $(f, n) \in ?F$
then obtain $s \text{ } t$ where $st: (s, t) \in DP\text{-simple } D \text{ } R$ and $(f, n) \in \text{funas-rule } (s, t)$
unfolding funas-trs-def by auto
then obtain u where $fn: (f, n) \in \text{funas-term } u$ and $u: u = s \vee u = t$ unfolding
 funas-rule-def
by auto
**from $st[\text{unfolded } DP\text{-simple-def}]$ obtain $l \text{ } r \text{ } uu$ where $lr: (l, r) \in R$ and $s: s =$
 $\# l$ **and $t: t = \# uu$ and $uu: r \supseteq uu$**
by force
**from $fn \text{ } u[\text{unfolded } s \text{ } t]$ obtain v where $fn: (f, n) \in \text{funas-term } (\# v)$ and $v: v$
 $= l \vee v = uu$ **by auto**
from fn have $fn: (f, n) \in \text{funas-term } v \cup \# (\text{funas-term } v)$
by $(\text{cases } v, \text{auto})$
from uu obtain C where $r: r = C \langle uu \rangle ..$
have $\text{funas-term } uu \subseteq \text{funas-term } r$ unfolding r by simp
with v have $\text{funas-term } v \subseteq \text{funas-rule } (l, r)$ unfolding funas-rule-def by auto
then have $\text{subset: } \text{funas-term } v \subseteq \text{funas-trs } R$ using lr unfolding funas-trs-def
by auto
with fn show $(f, n) \in ?H \cup ?I$ by auto
qed****

lemma $\text{funas-}DP\text{-on-subset}:$

$\text{funas-trs } (DP\text{-on } F \text{ } R) \subseteq \text{funas-trs } R \cup \# (\text{funas-trs } R)$

by (*rule order.trans* [*OF* - *funas-DP-simple-subset* [*of F*]])
 (*insert DP-on-subset-DP-simple*, *auto simp: funas-trs-def*)

end

end

theory *Q-Restricted-Rewriting*
imports *Sharp-Syntax*
begin

4.1 Q-Restricted Rewriting

definition *NF-subst* :: *bool* \Rightarrow (*'f*, *'v*) *rule* \Rightarrow (*'f*, *'v*) *subst* \Rightarrow (*'f*, *'v*) *terms* \Rightarrow *bool*

where

NF-subst *b* *lr* σ *Q* \longleftrightarrow (*b* \longrightarrow σ ' *vars-rule* *lr* \subseteq *NF-terms* *Q*)

lemma *NF-subst-False*[*simp*]: *NF-subst* *False* *lr* σ *Q* = *True*
unfolding *NF-subst-def* **by** *simp*

lemma *NF-subst-Empty*[*simp*]: *NF-subst* *nfs* *lr* σ {} = *True*
unfolding *NF-subst-def* **by** *simp*

lemma *NF-subst-right*: *nfs* \Longrightarrow *NF-subst* *nfs* (*s*, *t*) σ *Q* \Longrightarrow σ ' *vars-term* *t* \subseteq *NF-terms* *Q*
unfolding *NF-subst-def* *vars-rule-def* **by** *auto*

lemma *NF-substI*[*intro*]: **assumes** $\bigwedge x . \textit{nfs} \Longrightarrow x \in \textit{vars-term } l \vee x \in \textit{vars-term } r \Longrightarrow \sigma x \in \textit{NF-terms } Q$

shows *NF-subst* *nfs* (*l*, *r*) σ *Q*

unfolding *NF-subst-def*

proof (*intro impI allI subsetI*)

fix *t*

assume *nfs* **and** *t*: *t* $\in \sigma$ ' (*vars-rule* (*l*, *r*))

then obtain *x* **where** *t*: *t* = σx **and** *x*: *x* $\in \textit{vars-rule } (l, r)$ **by** *auto*

from *x* *assms*[*OF* $\langle \textit{nfs} \rangle$, *of x*] **show** *t* $\in \textit{NF-terms } Q$ **unfolding** *t*

unfolding *vars-rule-def* **by** *simp*

qed

inductive-set

qrstep :: *bool* \Rightarrow (*'f*, *'v*) *terms* \Rightarrow (*'f*, *'v*) *trs* \Rightarrow (*'f*, *'v*) *term rel*

for *nfs* **and** *Q* **and** *R*

where

subst[*intro*]: $\forall u \triangleleft s . \sigma . u \in \textit{NF-terms } Q \Longrightarrow (s, t) \in R \Longrightarrow \textit{NF-subst } \textit{nfs } (s, t) \sigma$
 $Q \Longrightarrow (s \cdot \sigma, t \cdot \sigma) \in \textit{qrstep } \textit{nfs } Q R \mid$

ctxt[*intro*]: $(s, t) \in \textit{qrstep } \textit{nfs } Q R \Longrightarrow (C \langle s \rangle, C \langle t \rangle) \in \textit{qrstep } \textit{nfs } Q R$

hide-fact (open)

qrstep.ctxst qrstep.subst
qrstepp.ctxst qrstepp.subst

lemma *qrstep-id[intro]*: $\forall u \triangleleft s. u \in \text{NF-terms } Q \implies (s, t) \in R \implies \text{NF-subst nfs } (s, t) \text{ Var } Q \implies (s, t) \in \text{qrstep nfs } Q R$
using *qrstep.subst[of s Var Q t R nfs]* **by** *auto*

lemma *supteq-qrstep-subset*:

$\{\triangleright\} O \text{ qrstep nfs } Q R \subseteq \text{qrstep nfs } Q R O \{\triangleright\}$
(is *?lhs* \subseteq *?rhs* **)**

proof

fix *s t*
assume $(s, t) \in ?lhs$
then obtain *u* **where** $s \triangleright u$ **and** $(u, t) \in \text{qrstep nfs } Q R$ **by** *auto*
from $\langle s \triangleright u \rangle$ **obtain** *C* **where** $s = C \langle u \rangle$ **by** *auto*
from *qrstep.ctxst[OF* $\langle (u, t) \in \text{qrstep nfs } Q R \rangle$ **]**
have $(s, C \langle t \rangle) \in \text{qrstep nfs } Q R$ **by** *(simp add: s)*
moreover have $C \langle t \rangle \triangleright t$ **by** *simp*
ultimately show $(s, t) \in ?rhs$ **by** *auto*

qed

definition

qrstep-r-p-s ::
 $\text{bool} \Rightarrow ('f, 'v) \text{ terms} \Rightarrow ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ rule} \Rightarrow \text{pos} \Rightarrow ('f, 'v) \text{ subst} \Rightarrow$
 $('f, 'v) \text{ trs}$
where
 $\text{qrstep-r-p-s nfs } Q R r p \sigma \equiv \{(s, t). \\$
 $(\forall u \triangleleft \text{fst } r \cdot \sigma. u \in \text{NF-terms } Q) \wedge \\$
 $p \in \text{poss } s \wedge r \in R \wedge s \mid - p = \text{fst } r \cdot \sigma \wedge t = \text{replace-at } s p (\text{snd } r \cdot \sigma) \wedge \\$
 $\text{NF-subst nfs } r \sigma Q\}$

definition

irstep :: $\text{bool} \Rightarrow ('f, 'v) \text{ trs} \Rightarrow ('f, 'v) \text{ term rel}$
where
 $\text{irstep nfs } R \equiv \text{qrstep nfs } (\text{lhss } R) R$

lemma *qrstep-rstep-conv[simp]*: $\text{qrstep nfs } \{ \} R = \text{rstep } R$

proof (*intro equalityI subrelI*)

fix *s t*
assume $(s, t) \in \text{rstep } R$
then obtain *C l r σ* **where** $s = C \langle l \cdot \sigma \rangle$ **and** $t = C \langle r \cdot \sigma \rangle$
and *lr*: $(l, r) \in R$ **by** *auto*
show $(s, t) \in \text{qrstep nfs } \{ \} R$ **unfolding** *s t*
by *(rule qrstep.ctxst[OF qrstep.subst[OF - lr]], auto)*
next
fix *s t* **assume** $(s, t) \in \text{qrstep nfs } \{ \} R$ **then show** $(s, t) \in \text{rstep } R$

by (induct) auto
qed

lemma *qrstep-trancl-ctxt*:
 assumes $(s, t) \in (\text{qrstep nfs } Q \ R)^+$
 shows $(C\langle s \rangle, C\langle t \rangle) \in (\text{qrstep nfs } Q \ R)^+$
 using *assms* by (induct) (auto intro: trancl-into-trancl)

The inductive definition really corresponds to the intuitive definition of Q-restricted rewriting.

lemma *qrstepE'*:
 assumes $(s, t) \in \text{qrstep nfs } Q \ R$
 shows $\exists C \ \sigma \ l \ r. (\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q) \wedge (l, r) \in R \wedge s = C\langle l \cdot \sigma \rangle \wedge t = C\langle r \cdot \sigma \rangle \wedge \text{NF-subst nfs } (l, r) \ \sigma \ Q$
 using *assms* **proof** (induct rule: *qrstep.induct*)
 case (ctxt $s \ t \ C$)
 then obtain $D \ \sigma \ l \ r$ where $\text{nf}: \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$
 and $lr: (l, r) \in R$ and $s: s = D\langle l \cdot \sigma \rangle$ and $t: t = D\langle r \cdot \sigma \rangle$
 and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$ by auto
 let $?C = C \circ_c D$
 have $s: C\langle s \rangle = ?C\langle l \cdot \sigma \rangle$ by (simp add: s)
 have $t: C\langle t \rangle = ?C\langle r \cdot \sigma \rangle$ by (simp add: t)
 show ?case using $\text{nf } lr \ s \ t \ \text{nfs}$ by blast
 next
 case (subst $s \ \sigma \ t$)
 show ?case
 apply (rule *exI[of - Hole]*)
 using *subst* by auto
 qed

lemma *qrstepE[elim]*:
 assumes $(s, t) \in \text{qrstep nfs } Q \ R$
 and $\bigwedge C \ \sigma \ l \ r. [\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q; (l, r) \in R; s = C\langle l \cdot \sigma \rangle; t = C\langle r \cdot \sigma \rangle; \text{NF-subst nfs } (l, r) \ \sigma \ Q] \implies P$
 shows P
 using *qrstepE'[of s t nfs Q R]* and *assms* by auto

lemma *qrstepI[intro]*:
 assumes $\text{nf}: \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$
 and $lr: (l, r) \in R$
 and $s: s = C\langle l \cdot \sigma \rangle$
 and $t: t = C\langle r \cdot \sigma \rangle$
 and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$
 shows $(s, t) \in \text{qrstep nfs } Q \ R$
 unfolding $s \ t$
 by (rule *qrstep.ctx[OF qrstep.subst[OF nf lr nfs]]*)

Every Q-step takes place at a specific position and using a specific rule and specific substitution.

lemma *qrstep-qrstep-r-p-s-conv*:

$(s, t) \in \text{qrstep nfs } Q \ R \iff (\exists r \ p \ \sigma. (s, t) \in \text{qrstep-r-p-s nfs } Q \ R \ r \ p \ \sigma)$

proof

assume $\exists r \ p \ \sigma. (s, t) \in \text{qrstep-r-p-s nfs } Q \ R \ r \ p \ \sigma$

then obtain $l \ r \ p \ \sigma$ **where** *NF-terms*: $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$

and $p: p \in \text{poss } s$ **and** $lr: (l, r) \in R$

and $s: s \mid - p = l \cdot \sigma$

and $t: t = \text{replace-at } s \ p \ (r \cdot \sigma)$

and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$

unfolding *qrstep-r-p-s-def* **by** *auto*

from *ctxt-supt-id*[*OF p*] **have** $s: s = (\text{ctxt-of-pos-term } p \ s) \langle l \cdot \sigma \rangle$ **unfolding** s
by *simp*

from $s \ t \ \text{NF-terms } lr \ \text{nfs}$ **show** $(s, t) \in \text{qrstep nfs } Q \ R$ **by** *auto*

next

assume $(s, t) \in \text{qrstep nfs } Q \ R$

then obtain $C \ l \ r \ \sigma$ **where** $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$

and $(l, r) \in R$ **and** $s: s = C \langle l \cdot \sigma \rangle$ **and** $t: t = C \langle r \cdot \sigma \rangle$

and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$

by *auto*

let $?p = \text{hole-pos } C$

have $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$ **by** *fact*

moreover have $?p \in \text{poss } s$ **unfolding** s **by** *simp*

moreover have $(l, r) \in R$ **by** *fact*

moreover have $s \mid - ?p = l \cdot \sigma$ **by** (*simp add: s*)

moreover have $t = \text{replace-at } s \ ?p \ (r \cdot \sigma)$ **by** (*simp add: s t*)

ultimately have $(s, t) \in \text{qrstep-r-p-s nfs } Q \ R \ (l, r) \ ?p \ \sigma$

by (*simp add: qrstep-r-p-s-def nfs*)

then show $\exists r \ p \ \sigma. (s, t) \in \text{qrstep-r-p-s nfs } Q \ R \ r \ p \ \sigma$ **by** *auto*

qed

lemma *qrstep-induct*[*case-names IH, induct set: qrstep*]:

assumes $(s, t) \in \text{qrstep nfs } Q \ R$

and *IH*: $\bigwedge C \ \sigma \ l \ r. \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q \implies (l, r) \in R \implies \text{NF-subst nfs } (l, r) \ \sigma \ Q \implies P \ C \langle l \cdot \sigma \rangle \ C \langle r \cdot \sigma \rangle$

shows $P \ s \ t$

proof –

from $\langle (s, t) \in \text{qrstep nfs } Q \ R \rangle$ **obtain** $C \ \sigma \ l \ r$ **where** *NF*: $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$

and $(l, r) \in R$ **and** $s: s = C \langle l \cdot \sigma \rangle$ **and** $t: t = C \langle r \cdot \sigma \rangle$

and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$

by *auto*

from *IH*[*OF NF* $\langle (l, r) \in R \rangle \ \text{nfs}$] **show** *?thesis* **unfolding** $s \ t$.

qed

lemma *qrstep-rule-conv*: $((s, t) \in \text{qrstep nfs } Q \ R) = (\exists \ lr \in R. (s, t) \in \text{qrstep nfs } Q \ \{lr\})$ (**is** $?l = ?r$)

proof

assume $?r$ **then show** $?l$ **by** *auto*

next

assume ?l **then show** ?r **by** blast
qed

lemma qrstep-empty-r[simp]: qrstep nfs Q {} = {}
using qrstep-rule-conv[of - - nfs Q {}] **by** auto

lemma qrstep-union: qrstep nfs Q (R \cup R') = qrstep nfs Q R \cup qrstep nfs Q R'
using qrstep-rule-conv[of - - nfs Q R]
qrstep-rule-conv[of - - nfs Q R']
qrstep-rule-conv[of - - nfs Q R \cup R']
by auto

lemma qrstep-all-mono: **assumes** R: R \subseteq R' **and** Q: NF-terms Q \subseteq NF-terms Q' **and** n: Q \neq {} \implies nfs' \implies nfs
shows qrstep nfs Q R \subseteq qrstep nfs' Q' R'
proof
fix s t **assume** (s, t) \in qrstep nfs Q R
then obtain C σ l r **where** $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$ **and** (l, r) \in R
and s = C(l \cdot σ) **and** t = C(r \cdot σ) **and** nfs: NF-subst nfs (l,r) σ Q **by** auto
moreover with n Q R **have** $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q'$
and NF-subst nfs' (l,r) σ Q' **and** (l,r) \in R' **unfolding** NF-subst-def **by** auto
ultimately show (s, t) \in qrstep nfs' Q' R' **by** auto
qed

lemma qrstep-rules-mono:
assumes R \subseteq R' **shows** qrstep nfs Q R \subseteq qrstep nfs Q R'
by (rule qrstep-all-mono[OF assms], auto)

lemma qrstep-mono:
assumes 1: R \subseteq R' **and** 2: NF-terms Q \subseteq NF-terms Q'
shows qrstep nfs Q R \subseteq qrstep nfs Q' R'
by (rule qrstep-all-mono[OF 1 2])

lemma qrstep-NF-anti-mono:
assumes Q \subseteq Q' **shows** qrstep nfs Q' R \subseteq qrstep nfs Q R
by (rule qrstep-mono[OF subset-refl NF-terms-anti-mono[OF assms]])

lemma qrstep-Id: qrstep nfs Q Id \subseteq Id
proof –
have qrstep nfs Q Id \subseteq qrstep nfs {} Id
by (rule qrstep-mono, auto)
also have ... \subseteq Id **using** rstep-id **by** auto
finally show ?thesis **by** auto
qed

lemma NF-terms-subset-criterion:
Q' \cap NF-terms Q = {} \longleftrightarrow NF-terms Q \subseteq NF-terms Q' (**is** ?lhs = ?rhs)

```

proof
  assume ?rhs then show ?lhs
proof (rule contrapos-pp)
  assume  $\neg$  ?lhs
  then obtain  $q$  where  $q \in Q'$  and  $q \in \text{NF-terms } Q$  by auto
  from  $\langle q \in Q' \rangle$  have  $q \notin \text{NF-terms } Q'$  by auto
  with  $\langle q \in \text{NF-terms } Q \rangle$  show  $\neg \text{NF-terms } Q \subseteq \text{NF-terms } Q'$  by blast
qed
next
assume ?lhs then show ?rhs
proof (rule contrapos-pp)
  assume  $\neg$  ?rhs
  then obtain  $t$  where  $\text{NF}: t \in \text{NF-terms } Q$  and  $t \notin \text{NF-terms } Q'$  by auto
  then obtain  $s$  where  $(t, s) \in \text{rstep } (\text{Id-on } Q')$  by (auto simp: NF-def)
  then obtain  $C \ q \ \sigma$  where  $q \in Q'$  and  $t: t = C \langle q \cdot \sigma \rangle$  by auto
  have  $q \in \text{NF-terms } Q$ 
  proof
    fix  $D \ l \ \tau$ 
    assume  $q: q = D \langle l \cdot \tau \rangle$  and  $l: l \in Q$ 
    have  $(t, t) \in \text{rstep } (\text{Id-on } Q)$  unfolding  $t \ q$ 
      by (rule rstepI[of  $l \ l \ - \ C \circ_c (D \cdot_c \sigma) \ \tau \circ_s \sigma$ ], insert  $l$ , auto)
    then have  $t \notin \text{NF-terms } Q$  by auto
    with  $\langle t \in \text{NF-terms } Q \rangle$  show False by blast
  qed
  with  $\langle q \in Q' \rangle$  show  $Q' \cap \text{NF-terms } Q \neq \{\}$  by auto
qed
qed

lemma qrstep-subset-rstep[intro,simp]:  $\text{qrstep nfs } Q \ R \subseteq \text{rstep } R$ 
  by (simp only: qrstep-rstep-conv[symmetric, of  $R \ \text{nfs}$ ], rule qrstep-NF-anti-mono, auto)

lemma qrstep-into-rstep:  $(s, t) \in \text{qrstep nfs } Q \ R \implies (s, t) \in \text{rstep } R$ 
  using qrstep-subset-rstep by auto

lemma qrsteps-into-rsteps:  $(s, t) \in (\text{qrstep nfs } Q \ R)^* \implies (s, t) \in (\text{rstep } R)^*$ 
  using rtrancl-mono[OF qrstep-subset-rstep] by auto

lemma qrstep-preserves-funus-terms:
  assumes  $r: \text{funas-term } r \subseteq F$ 
  and  $sF: \text{funas-term } s \subseteq F$  and  $\text{step}: (s, t) \in \text{qrstep nfs } Q \ \{(l, r)\}$  and  $\text{vars}: \text{vars-term } r \subseteq \text{vars-term } l$ 
  shows  $\text{funas-term } t \subseteq F$ 
proof –
  from  $\text{step}$  obtain  $C \ \sigma$  where
     $s: s = C \langle l \cdot \sigma \rangle$  and  $t: t = C \langle r \cdot \sigma \rangle$  by auto
  have  $\text{fs}: \text{funas-term } s = \text{funas-ctxt } C \cup \text{funas-term } l \cup \bigcup (\text{funas-term } \sigma \text{ ` } \text{vars-term } l)$  unfolding  $s$  using funas-term-subst by auto
  have  $\text{funas-term } t = \text{funas-ctxt } C \cup \text{funas-term } r \cup \bigcup (\text{funas-term } \sigma \text{ ` } \text{vars-term } r)$ 

```

r) **unfolding** t **using** *funas-term-subst* **by** *auto*
 then **have** *funas-term* $t \subseteq \text{funas-ctxt } C \cup \text{funas-term } r \cup \bigcup (\text{funas-term } \sigma \text{ 'vars-term } l)$ **using** $\langle \text{vars-term } r \subseteq \text{vars-term } l \rangle$ **by** *auto*
 with $\langle \text{funas-term } r \subseteq F \rangle \langle \text{funas-term } s \subseteq F \rangle$
 show *?thesis* **unfolding** fs **by** *force*
qed

lemma *ctxt-of-pos-term-grstep-below*:
 assumes *step*: $(s, t) \in \text{grstep-r-p-s nfs } Q \ R \ r \ p' \ \sigma$ **and** *le*: $p \leq_p p'$
 shows *ctxt-of-pos-term* $p \ s = \text{ctxt-of-pos-term } p \ t$
proof –
 from *step*[*unfolded grstep-r-p-s-def*] **have** $p': p' \in \text{poss } s$ **and** $t: t = \text{replace-at } s \ p' \ (\text{snd } r \cdot \sigma)$ **by** *auto*
 show *?thesis* **unfolding** t
proof (*rule ctxt-of-pos-term-replace-at-below*[*OF - le, of s, symmetric*])
 from $p' \leq p$ **show** $p \in \text{poss } s$ **unfolding** *less-eq-pos-def* **by** *auto*
qed
qed

lemma *parallel-grstep-subt-at*:
 fixes $p :: \text{pos}$
 assumes *step*: $(s, t) \in \text{grstep-r-p-s nfs } Q \ R \ lr \ p \ \sigma$
 and *par*: $p \perp q$
 and $q: q \in \text{poss } s$
 shows $s \mid - q = t \mid - q \wedge q \in \text{poss } t$
proof –
 note *step* = *step*[*unfolded grstep-r-p-s-def*]
 from *step* **have** $t: t = \text{replace-at } s \ p \ (\text{snd } lr \cdot \sigma)$ **and** $p: p \in \text{poss } s$ **by** *auto*
 show *?thesis* **unfolding** t
 by (*rule conjI, rule parallel-replace-at-subt-at*[*symmetric, OF par p q*], *insert parallel-poss-replace-at*[*OF par p*] q , *auto*)
qed

lemma *grstep-subt-at-gen*:
 assumes *step*: $(s, t) \in \text{grstep-r-p-s nfs } Q \ R \ lr \ (p @ q) \ \sigma$
 shows $(s \mid - p, t \mid - p) \in \text{grstep-r-p-s nfs } Q \ R \ lr \ q \ \sigma$
proof –
 from *step*[*unfolded grstep-r-p-s-def*]
 have *NF*: $\forall u. u \triangleleft \text{fst } lr \cdot \sigma. u \in \text{NF-terms } Q$
 and $pq: p @ q \in \text{poss } s$
 and $lr: lr \in R$
 and $s: s \mid - (p @ q) = \text{fst } lr \cdot \sigma$
 and $t: t = \text{replace-at } s \ (p @ q) \ (\text{snd } lr \cdot \sigma)$
 and *nfs*: *NF-subst nfs* $lr \ \sigma \ Q$
 by *auto*
 from *pq*[*unfolded poss-append-poss*] **have** $p: p \in \text{poss } s$ **and** $q: q \in \text{poss } (s \mid - p)$
 by *auto*

```

show ?thesis
  unfolding qrstep-r-p-s-def
  apply (intro CollectI, unfold split)
  apply (intro conjI NF q lr nfs)
  subgoal by (rule s[unfolded subt-at-append[OF p]])
  subgoal by (unfold t ctxt-of-pos-term-append[OF p], simp add: replace-at-subt-at[OF
p])
  done
qed

```

```

lemma qrstep-r-p-s-imp-poss:
  assumes step: (s,t) ∈ qrstep-r-p-s nfs Q R lr p σ
  shows p ∈ poss s ∧ p ∈ poss t
proof −
  from step[unfolded qrstep-r-p-s-def]
  have ps: p ∈ poss s
    and t: t = replace-at s p (snd lr · σ)
    by auto
  have pt: p ∈ poss t unfolding t
    using hole-pos-ctxt-of-pos-term[OF ps]
    hole-pos-poss[of ctxt-of-pos-term p s] by auto
  with ps show ?thesis by auto
qed

```

```

lemma qrstep-subt-at:
  assumes step: (s, t) ∈ qrstep-r-p-s nfs Q R lr p σ
  shows (s |- p, t |- p) ∈ qrstep-r-p-s nfs Q R lr [] σ
  by (rule qrstep-subt-at-gen, insert step, simp)

```

```

lemma parallel-qrstep-poss:
  fixes q :: pos
  assumes par: q ⊥ p
    and q: q ∈ poss s
    and step: (s, t) ∈ qrstep-r-p-s nfs Q R r p σ
  shows q ∈ poss t
proof −
  from step[unfolded qrstep-r-p-s-def]
  have t: t = replace-at s p (snd r · σ) and p: p ∈ poss s by auto
  from parallel-poss-replace-at[OF parallel-pos-sym[OF par] p]
  show ?thesis unfolding t using q by simp
qed

```

```

lemma parallel-qrstep-ctxt-to-term-list:
  fixes q :: pos
  assumes par: q ⊥ p
    and poss: q ∈ poss s

```

and step: $(s, t) \in \text{qrstep-r-p-s nfs } Q R r p \sigma$
shows $\exists p'. p' \leq_p p \wedge (\exists i. i < \text{length } (\text{ctxt-to-term-list } (\text{ctxt-of-pos-term } q s)))$
 \wedge
 $\text{ctxt-to-term-list } (\text{ctxt-of-pos-term } q s) ! i = s \mid - p' \wedge$
 $\text{ctxt-to-term-list } (\text{ctxt-of-pos-term } q t) =$
 $(\text{ctxt-to-term-list } (\text{ctxt-of-pos-term } q s))[i := t \mid - p']$
proof –
from $\text{parallel-remove-prefix}[OF \text{ par}]$ **obtain** $pp \ i1 \ i2 \ q1 \ q2$ **where**
 $q: q = pp @ i1 \# q1$ **and** $p: p = pp @ i2 \# q2$ **and** $i12: i1 \neq i2$
by *blast*
let $?p1 = i1 \# q1$
let $?p2 = i2 \# q2$
let $?c = \text{ctxt-of-pos-term}$
let $?cc = \lambda p \ t. \text{ctxt-to-term-list } (?c \ p \ t)$
note $qr\text{-def} = \text{qrstep-r-p-s-def}$
show $?thesis$ **unfolding** $p \ q$
proof (*intro exI, intro conjI*)
show $pp @ [i2] \leq_p pp @ ?p2$ **unfolding** less-eq-pos-def **by** *simp*
next
show $\exists i < \text{length } (?cc \ (pp @ ?p1) \ s).$
 $?cc \ (pp @ ?p1) \ s ! i = s \mid - (pp @ [i2]) \wedge$
 $?cc \ (pp @ ?p1) \ t = (?cc \ (pp @ ?p1) \ s) [i := t \mid - (pp @ [i2])]$
using $\text{step poss unfolding } p \ q$
proof (*induct pp arbitrary: s t*)
case ($\text{Cons } i \ p \ s \ t$)
from $\text{Cons}(2)$ **have** $\text{step: } (s, t) \in \text{qrstep-r-p-s nfs } Q R r ([i] @ (p @ ?p2)) \sigma$
by *simp*
note $\text{istep} = \text{qrstep-subt-at-gen}[OF \text{ step}]$
note $\text{step} = \text{step}[\text{unfolded } qr\text{-def}]$
from step **have** $p2: i \# p @ ?p2 \in \text{poss } s$ **by** *auto*
then obtain $f \ ss$ **where** $s: s = \text{Fun } f \ ss$ **by** (*cases s, auto*)
with $p2$ **have** $\text{iss: } i < \text{length } ss$ **by** *auto*
from $p2[\text{unfolded } s]$ **have** $p2i: p @ [i2] \in \text{poss } (ss ! i)$ **by** *simp*
from $p2i \ \text{iss}$ **have** $ip2: i \# p \in \text{poss } s$ **unfolding** s **by** *simp*
from iss **have** $si: s \mid - [i] = ss ! i$ **unfolding** s **by** *simp*
from $\text{Cons}(3)$ **have** $p @ ?p1 \in \text{poss } (s \mid - [i])$ **unfolding** s **using** iss **by** *simp*
from $\text{Cons}(1)[OF \text{ istep this}]$
obtain i' **where** $i': i' < \text{length } (?cc \ (p @ ?p1) \ (s \mid - [i]))$ **and**
 $\text{id1: } ?cc \ (p @ ?p1) \ (s \mid - [i]) ! i' = s \mid - [i] \mid - (p @ [i2])$ **and**
 $\text{id2: } ?cc \ (p @ ?p1) \ (t \mid - [i]) = (?cc \ (p @ ?p1) \ (s \mid - [i]))[i' := t \mid - [i]]$
 $\mid - (p @ [i2])$ **by** *blast*
from step **have** $tp2: t = \text{replace-at } s \ (i \# p @ ?p2) \ (\text{snd } r \cdot \sigma)$ **by** *simp*
have $\text{ipless: } i \# (p @ [i2]) \leq_p i \# p @ ?p2$ **by** *simp*
have $\text{ipt: } i \# (p @ [i2]) \in \text{poss } t$ **unfolding** $tp2$
by (*rule replace-at-below-poss[OF p2 ipless]*)
have $t = \text{replace-at } t \ (i \# (p @ [i2])) \ (t \mid - (i \# (p @ [i2])))$
by (*rule ctxt-supt-id[symmetric, OF ipt]*)
also have $\dots = \text{replace-at } (\text{Fun } f \ ss) \ (i \# (p @ [i2])) \ (t \mid - (i \# (p @ [i2])))$ (*is - = ?t*)

```

    unfolding s[symmetric]
    using ctxt-of-pos-term-qrstep-below[OF Cons(2), of i # (p @ [i2])] ipless
  by auto
    finally have t: t = ?t .
    have t |- [i] = ?t |- [i] using t by simp
    also have ... = replace-at (ss ! i) (p @ [i2]) (t |- (i # (p @ [i2]))) (is - = ?ti)
      using iss by (simp add: nth-append)
    finally have ti: t |- [i] = ?ti .
    have cs: ?c ((i # p) @ ?p2) s = More f (take i ss) (?c (p @ ?p2) (ss ! i))
      (drop (Suc i) ss)
    unfolding s by simp
    have i'': i' < length (?cc ((i # p) @ ?p1) s) using i' unfolding s si by simp
    show ?case
    proof (rule exI[of - i'], intro conjI, rule i'')
      show ?cc ((i # p) @ ?p1) s ! i' = s |- ((i # p) @ [i2])
        unfolding s using si id1 i' by (simp add: nth-append)
    next
      from iss have min: min (length ss) i = i by simp
      note i' = i'[unfolded si]
      have i'': i' < length (?cc (p @ i1 # q1) (ss ! i) @ take i ss @ drop (Suc i)
ss)
        using i' by simp
      have ?cc ((i # p) @ ?p1) t = ?cc ((i # p) @ ?p1) ?t using t by simp
      also have ... = (?cc ((i # p) @ ?p1) s) [i' := t |- (i # (p @ [i2]))]
        unfolding s using min id2 i' iss si
      unfolding ti
      by (simp add: replace-at-subt-at[OF p2i] upd-conv-take-nth-drop[OF i'']
upd-conv-take-nth-drop[OF i'] nth-append)
      finally show ?cc ((i # p) @ i1 # q1) t = (?cc ((i # p) @ i1 # q1) s)
        [i' := t |- ((i # p) @ [i2])] by simp
    qed
  next
    case Nil
    have eps:  $\bigwedge p. [] @ p = p \bigwedge t. t |- [] = t$  by auto
    note Empty = Nil[unfolded eps]
    note step = Empty[unfolded qr-def]
    from step have p2: i2 # q2  $\in$  poss s by auto
    from Empty have p1: i1 # q1  $\in$  poss s by simp
    then obtain f ss where s: s = Fun f ss by (cases s, auto)
    from s p2 have i2: i2 < length ss by auto
    from s p1 have i1: i1 < length ss by auto
    from step have t: t = replace-at (Fun f ss) (i2 # q2) (snd r ·  $\sigma$ ) unfolding
s by simp
    have r $\sigma$ : snd r ·  $\sigma$  = t |- (i2 # q2) unfolding t
      by (rule replace-at-subt-at[symmetric], insert p2, unfold s)
    have tt: t = replace-at (Fun f ss) (i2 # q2) (t |- (i2 # q2)) (is t = ?t)
      unfolding r $\sigma$ [symmetric] unfolding t ..
    have t |- [i2] = ?t |- [i2] using tt by simp
    also have ... = replace-at (ss ! i2) q2 (t |- (i2 # q2)) (is - = ?ti)

```

```

    using i2 by (simp add: nth-append)
  finally have ti: t |- [i2] = ?ti .
  have i2ss: take i2 ss @ ss ! i2 # drop (Suc i2) ss = ss
    using upd-conv-take-nth-drop[OF i2, symmetric] list-update-id by auto
  let ?n = length (?cc q1 (ss ! i1))
  from i1 have min1: min (length ss) i1 = i1 by simp
  from i2 have min2: min (length ss) i2 = i2 by simp
  from i12 have i12: i1 < i2 ∨ i2 < i1 by auto
  obtain i2' where i2': i2' = (if i1 < i2 then i2 - Suc 0 else i2) by auto
  {
    from i12
    have (take i1 ss @ drop (Suc i1) ss) ! i2' = ss ! i2
    proof
      assume i2 < i1
      then show ?thesis by (auto simp: nth-append i2' i2)
    next
      assume i12: i1 < i2
      then have i2 = Suc i1 + (i2 - Suc i1) by simp
      then obtain k where i2k: i2 = Suc i1 + k by blast
      from i2[unfolded i2k] have Suc i1 + k ≤ length ss by simp
      then show ?thesis
        by (metis (no-types, lifting) Suc-less-eq Suc-pred i2k add-gr-0 add-leD1
diff-Suc-Suc
diff-add-inverse i12 i2' length-take min1 not-add-less1 nth-append
nth-drop zero-less-Suc)
    qed
  } note i2'id = this
  from tt have tt: ∧ p. ?cc p t = ?cc p ?t by simp
  show ?case unfolding eps s ti unfolding tt
  proof (rule exI[of - ?n + i2'], intro conjI)
    show ?cc (i1 # q1) (Fun f ss) ! (?n + i2') = Fun f ss |- [i2]
      by (simp add: i2'id)
  next
    from i1 have len: i1 + (length ss - Suc i1) = length ss - Suc 0 by auto
    from i12 have
      i2'len: i2' < length ss - Suc 0 unfolding i2' using i1 i2 by auto
    show length (?cc q1 (ss ! i1)) + i2' < length (?cc (i1 # q1) (Fun f ss))
      by (auto simp: i2 i1 min1 len i2'len)
  next
    from i12
    show ?cc (i1 # q1) ?t = (?cc (i1 # q1) (Fun f ss)) [?n + i2' := ?ti]
    proof
      assume i12: i2 < i1
      then have len: ?n + i2' < length (?cc (i1 # q1) (Fun f ss))
        unfolding i2' using min1 i1 i2 by simp
      from i12 have min12: min i2 i1 = i2 min i1 i2 = i2 by auto
      have drop: drop (Suc i2) ss ! (i1 - Suc i2) = ss ! i1 using i12 i1 by
simp
      from i12 have i1 = Suc i2 + (i1 - Suc i2) by simp

```

```

then obtain  $k$  where  $i1k$ :  $i1 = \text{Suc } i2 + k$  by blast
show ?thesis using  $i12$ 
  unfolding upd-conv-take-nth-drop[OF len]
  unfolding  $i2'$ 
  by (simp add: min12  $i1$   $i2$  min1 min2 nth-append drop, simp add:  $i1k$ 
ac-simps take-drop)
next
  assume  $i12$ :  $i1 < i2$ 
  then have len:  $?n + i2' < \text{length } (?cc (i1 \# q1) (\text{Fun } f \text{ ss}))$ 
  unfolding  $i2'$  using min1  $i1$   $i2$  by simp
  from  $i12$  have min12:  $\min i2 i1 = i1$   $\min i1 i2 = i1$  by auto
  from  $i12$  have  $i2 = \text{Suc } i1 + (i2 - \text{Suc } i1)$  by simp
  then obtain  $k$  where  $i2k$ :  $i2 = \text{Suc } i1 + k$  by blast
  have minik:  $\min i1 (i1 + k) = i1$  by simp
  show ?thesis using  $i12$ 
    unfolding upd-conv-take-nth-drop[OF len]
    unfolding  $i2'$ 
    by (simp add: min12  $i1$   $i2$  min1 min2 nth-append, simp add:  $i2k$  take-drop
ac-simps minik)
qed
qed
qed
qed
qed

```

```

lemma parallel-grstep:
  fixes  $p1 :: pos$ 
  assumes  $p12$ :  $p1 \perp p2$ 
  and  $p1$ :  $p1 \in \text{poss } t$ 
  and  $p2$ :  $p2 \in \text{poss } t$ 
  and step2:  $t \vdash p2 = l2 \cdot \sigma2 \ \forall \ u \triangleleft l2 \cdot \sigma2. u \in \text{NF-terms } Q \ (l2, r2) \in R$ 
  NF-subst nfs ( $l2, r2$ )  $\sigma2$   $Q$ 
  shows (replace-at  $t$   $p1$   $v$ , replace-at (replace-at  $t$   $p1$   $v$ )  $p2$  ( $r2 \cdot \sigma2$ ))  $\in$  grstep nfs
   $Q$   $R$  (is (?one, ?two)  $\in$  -)
proof -
  show ?thesis unfolding grstep-grstep-r-p-s-conv
proof (intro exI)
  show (?one, ?two)  $\in$  grstep-r-p-s nfs  $Q$   $R$  ( $l2, r2$ )  $p2$   $\sigma2$ 
  unfolding grstep-r-p-s-def
  apply (intro CollectI, unfold split fst-conv snd-conv)
  apply (unfold parallel-replace-at-subt-at[OF  $p12$   $p1$   $p2$ ])
  apply (unfold parallel-poss-replace-at[OF  $p12$   $p1$ ])
  apply (intro conjI step2(2) refl  $p2$ )
  by (insert step2, auto)
qed
qed

```

```

lemma parallel-grstep-swap:
  fixes  $p1 :: pos$ 

```

assumes $p12: p1 \perp p2$
and two: $(t, s) \in \text{qrstep-r-p-s nfs } Q1 \ R1 \ r1 \ p1 \ \sigma1 \ O \ \text{qrstep-r-p-s nfs } Q2 \ R2 \ r2 \ p2 \ \sigma2$
shows $(t, s) \in \text{qrstep-r-p-s nfs } Q2 \ R2 \ r2 \ p2 \ \sigma2 \ O \ \text{qrstep-r-p-s nfs } Q1 \ R1 \ r1 \ p1 \ \sigma1$
proof –
let $?R1 = \text{qrstep-r-p-s nfs } Q1 \ R1$
let $?R2 = \text{qrstep-r-p-s nfs } Q2 \ R2$
from two obtain u **where** $tu: (t, u) \in ?R1 \ r1 \ p1 \ \sigma1$ **and** $us: (u, s) \in ?R2 \ r2 \ p2 \ \sigma2$ **by auto**
from $tu[\text{unfolded qrstep-r-p-s-def}]$
have $\text{step1}: t \vdash p1 = \text{fst } r1 \cdot \sigma1 \ \forall \ u \triangleleft \text{fst } r1 \cdot \sigma1. u \in \text{NF-terms } Q1$ **and** $r1: r1 \in R1$
and $p1: p1 \in \text{poss } t$ **and** $u: u = \text{replace-at } t \ p1 \ (\text{snd } r1 \cdot \sigma1)$
and $\text{nfs1}: \text{NF-subst nfs } r1 \ \sigma1 \ Q1$ **by auto**
from $us[\text{unfolded qrstep-r-p-s-def, simplified}]$
have $\text{step2}: u \vdash p2 = \text{fst } r2 \cdot \sigma2 \ \forall \ u \triangleleft \text{fst } r2 \cdot \sigma2. u \in \text{NF-terms } Q2$ **and** $r2: r2 \in R2$
and $p2: p2 \in \text{poss } u$ **and** $s: s = \text{replace-at } u \ p2 \ (\text{snd } r2 \cdot \sigma2)$
and $\text{nfs2}: \text{NF-subst nfs } r2 \ \sigma2 \ Q2$ **by auto**
from parallel-poss-replace-at $[OF \ p12 \ p1] \ p2$ **have** $p2': p2 \in \text{poss } t$ **unfolding** u **by simp**
have one: $(t, \text{replace-at } t \ p2 \ (\text{snd } r2 \cdot \sigma2)) \in ?R2 \ r2 \ p2 \ \sigma2$ **(is** $(t, ?t) \in -$ **)**
unfolding qrstep-r-p-s-def
apply $(\text{intro CollectI, unfold split})$
apply $(\text{unfold step2}(1)[\text{symmetric}] \ u \ \text{parallel-replace-at-subt-at}[OF \ p12 \ p1 \ p2'])$
apply $(\text{intro conjI } p2' \ r2 \ \text{refl nfs2})$
by $(\text{metis } p1 \ p12 \ p2' \ \text{parallel-replace-at-subt-at step2}(1) \ \text{step2}(2) \ u)$
have $p21: p2 \perp p1$ **by** $(\text{rule parallel-pos-sym}[OF \ p12])$
have $p1': p1 \in \text{poss } ?t$ **using** $\text{parallel-poss-replace-at}[OF \ p21 \ p2'] \ p1$ **by simp**
have two: $(?t, \text{replace-at } ?t \ p1 \ (\text{snd } r1 \cdot \sigma1)) \in ?R1 \ r1 \ p1 \ \sigma1$ **(is** $(-, ?t') \in -$ **)**
unfolding qrstep-r-p-s-def
apply $(\text{intro CollectI, unfold split})$
apply $(\text{intro conjI } p1' \ r1 \ \text{nfs1 refl})$
subgoal by $(\text{rule step1}(2))$
subgoal by $(\text{unfold step1}(1)[\text{symmetric}] \ \text{parallel-replace-at-subt-at}[OF \ p21 \ p2'] \ p1], \text{simp})$
done
with one have steps: $(t, ?t') \in ?R2 \ r2 \ p2 \ \sigma2 \ O \ ?R1 \ r1 \ p1 \ \sigma1$ **by auto**
have $s = ?t'$ **unfolding** $s \ u$
by $(\text{rule parallel-replace-at}[OF \ p12 \ p1 \ p2'])$
then show $?thesis$ **using steps by auto**
qed

lemma *normalize-subterm-qrsteps-count:*

assumes $p: p \in \text{poss } t$
and steps: $(t, s) \in (\text{qrstep nfs } Q \ R) \sim^n$
and $s: s \in \text{NF-terms } Q$

shows $\exists n1\ n2\ u. (t \mid\!-\ p, u) \in (qrstep\ nfs\ Q\ R)^{\sim n1} \wedge u \in NF\text{-terms}\ Q \wedge$
 $(replace\text{-}at\ t\ p\ u, s) \in (qrstep\ nfs\ Q\ R)^{\sim n2} \wedge n = n1 + n2$
proof –
let $?Q = \lambda n\ n1\ n2\ t\ u\ s. (t \mid\!-\ p, u) \in (qrstep\ nfs\ Q\ R)^{\sim n1} \wedge u \in NF\text{-terms}\ Q$
 $\wedge (replace\text{-}at\ t\ p\ u, s) \in (qrstep\ nfs\ Q\ R)^{\sim n2} \wedge n = n1 + n2$
let $?P = \lambda n\ t\ s. (\exists n1\ n2\ u. ?Q\ n\ n1\ n2\ t\ u\ s)$
have $?P\ n\ t\ s$ **using** *steps s p*
proof (*induct n arbitrary: t s*)
case 0
then have $t: t \in NF\text{-terms}\ Q$ **and** $p: p \in poss\ t$ **and** $s: s = t$ **by** *auto*
show $\exists n1\ n2\ u. ?Q\ 0\ n1\ n2\ t\ u\ s$
proof (*rule exI[of - 0], rule exI[of - 0], rule exI[of - t \mid\!-\ p], intro conjI*)
from p **have** $t \mid\!-\ p \trianglelefteq t$ **by** (*rule subt-at-imp-supteq*)
from $NF\text{-subterm}[OF\ t\ this]$
show $t \mid\!-\ p \in NF\text{-terms}\ Q$.
next
have $replace\text{-}at\ t\ p\ (t \mid\!-\ p) = t$ **using** p **by** (*rule ctxt-supt-id*)
then show $(replace\text{-}at\ t\ p\ (t \mid\!-\ p), s) \in (qrstep\ nfs\ Q\ R)^{\sim 0}$ **unfolding** s **by**
simp
qed *auto*
next
case (*Suc n*)
then have $p: p \in poss\ t$ **by** *simp*
from $relpow\text{-}Suc\text{-}D2[OF\ Suc(2)]$ **obtain** u **where** $tu: (t, u) \in qrstep\ nfs\ Q\ R$
and $us: (u, s) \in qrstep\ nfs\ Q\ R^{\sim n}$ **by** *auto*
note $ind = Suc(1)[OF\ us\ Suc(3)]$
from $tu[unfolding\ qrstep\text{-}qrstep\text{-}r\text{-}p\text{-}s\text{-}conv]$
obtain $r\ p'\ \sigma$ **where** $tu': (t, u) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ r\ p'\ \sigma$ **by** *auto*
from $tu'[unfolding\ qrstep\text{-}r\text{-}p\text{-}s\text{-}def]$
have $NF: \forall u \triangleleft fst\ r \cdot \sigma. u \in NF\text{-terms}\ Q$ **and** $p': p' \in poss\ t$ **and** $r: r \in R$
and $t: t \mid\!-\ p' = fst\ r \cdot \sigma$
and $u: u = replace\text{-}at\ t\ p'\ (snd\ r \cdot \sigma)$
and $nfsr: NF\text{-subst}\ nfs\ r\ \sigma\ Q$ **by** *auto*
from *pos-cases* **have** *cases*: $p \leq_p p' \vee p' <_p p \vee p \perp p'$.
{
assume $pp': p \leq_p p'$
then obtain p'' **where** $p': p' = p @ p''$ **unfolding** *less-eq-pos-def* **by** *auto*
from $p'\ p''$ **have** $tp'': p'' \in poss\ (t \mid\!-\ p)$ **by** *simp*
from t **have** $t: t \mid\!-\ p \mid\!-\ p'' = fst\ r \cdot \sigma$ **unfolding** p'' *subt-at-append[OF p]* .
from $replace\text{-}at\text{-}below\text{-}poss[OF\ p'\ pp']$ **have** $pu: p \in poss\ u$ **unfolding** u .
from $ind[OF\ this]$ **obtain** $n1\ n2\ w$ **where** $steps1: (u \mid\!-\ p, w) \in qrstep\ nfs\ Q$
 $R^{\sim n1}$ **and** $w: w \in NF\text{-terms}\ Q$
and $steps2: (replace\text{-}at\ u\ p\ w, s) \in qrstep\ nfs\ Q\ R^{\sim n2}$ **and** $sum: n = n1$
 $+ n2$ **by** *auto*
have $tu: ctxt\text{-}of\text{-}pos\text{-}term\ p\ t = ctxt\text{-}of\text{-}pos\text{-}term\ p\ u$ **unfolding** u **by**
(simp add: ctxt-of-pos-term-replace-at-below[OF p pp'])
have $(t \mid\!-\ p, u \mid\!-\ p) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ r\ p''\ \sigma$ **unfolding** *qrstep-r-p-s-def*
apply (*intro CollectI, unfold split*)
apply (*intro conjI NF tp'' r t nfsr*)

by (unfold u p'' ctxt-of-pos-term-append[OF p], simp add: replace-at-subt-at[OF p])
 then have (t |- p, u |- p) ∈ grstep nfs Q R unfolding grstep-grstep-r-p-s-conv
 by blast
 from relpow-Suc-I2[OF this steps1] have steps1: (t |- p, w) ∈ grstep nfs Q
 R \sim Suc n1 .
 have ?case
 apply (intro exI conjI)
 apply (rule steps1)
 apply (rule w)
 apply (unfold tu, rule steps2)
 by (unfold sum, simp)
 } note above = this
 {
 assume p'p: p' <_p p
 from less-pos-imp-supt[OF p'p p] have t |- p < t |- p' .
 with NF[unfolded t[symmetric]] have tp: t |- p ∈ NF-terms Q by blast
 have steps1: (t |- p, t |- p) ∈ grstep nfs Q R \sim 0 by simp
 have ?case
 apply (intro exI conjI)
 apply (rule steps1)
 apply (rule tp)
 apply (unfold ctxt-supt-id[OF p], rule Suc(2))
 by simp
 } note below = this
 {
 assume pp': p ⊥ p'
 from parallel-pos-sym[OF this] have p'p: p' ⊥ p .
 from parallel-poss-replace-at[OF this p'] p have pu: p ∈ poss u unfolding u
 by blast
 from ind[OF this] obtain n1 n2 w where steps1: (u |- p, w) ∈ grstep nfs Q
 R \sim n1 and w: w ∈ NF-terms Q
 and steps2: (replace-at u p w, s) ∈ grstep nfs Q R \sim n2 and sum: n = n1
 + n2 by auto
 from parallel-replace-at-subt-at[OF p'p p' p] have uptp: u |- p = t |- p
 unfolding u .
 have ?case
 proof (intro exI conjI, rule steps1[unfolded uptp], rule w)
 show Suc n = n1 + Suc n2 unfolding sum by simp
 next
 from r have r: (fst r, snd r) ∈ R by simp
 have (replace-at t p w, replace-at u p w) ∈ grstep nfs Q R unfolding u
 unfolding parallel-replace-at[OF p'p p' p]
 by (rule parallel-grstep[OF pp' p p' t NF r], insert nfsr, auto)
 from relpow-Suc-I2[OF this steps2]
 show (replace-at t p w, s) ∈ grstep nfs Q R \sim (Suc n2) .
 } qed
 } note parallel = this
 from cases above below parallel show ?case by blast

qed
 then show ?thesis by auto
 qed

lemma *normalize-subterm-qrstps*:

assumes $p: p \in \text{poss } t$
 and $\text{steps}: (t, s) \in (\text{qrstep nfs } Q \ R)^*$
 and $s: s \in \text{NF-terms } Q$
 shows $\exists u. (t \mid\!-\ p, u) \in (\text{qrstep nfs } Q \ R)^* \wedge u \in \text{NF-terms } Q \wedge (\text{replace-at } t \ p \ u, s) \in (\text{qrstep nfs } Q \ R)^*$
proof –
 from *rtrancl-imp-relpow*[*OF steps*] obtain n where $\text{steps}: (t, s) \in \text{qrstep nfs } Q \ R \rightsquigarrow^n$ by auto
 from *normalize-subterm-qrstps-count*[*OF p steps s*]
 obtain $n1 \ n2 \ u$ where $\text{steps1}: (t \mid\!-\ p, u) \in \text{qrstep nfs } Q \ R \rightsquigarrow^{n1}$ and $u: u \in \text{NF-terms } Q$
 and $\text{steps2}: (\text{replace-at } t \ p \ u, s) \in \text{qrstep nfs } Q \ R \rightsquigarrow^{n2}$ by auto
 from *relpow-imp-rtrancl*[*OF steps1*] *relpow-imp-rtrancl*[*OF steps2*] u show ?thesis by blast
 qed

lemma *parallel-qrstps-r-p-s*:

fixes $p1 :: \text{pos}$
 assumes $p12: p1 \perp p2$
 and $p1: p1 \in \text{poss } t$
 and $\text{step}: (t, s) \in \text{qrstep-r-p-s nfs } Q \ R \text{ lr } p2 \ \sigma$
 shows $(\text{replace-at } t \ p1 \ w, \text{replace-at } s \ p1 \ w) \in \text{qrstep nfs } Q \ R \text{ (is } (?one, ?two) \in \text{ -})$
proof –
 note $d = \text{qrstep-r-p-s-def}$
 from *step*[*unfolded d*] have $\text{NF}: \forall u \triangleleft \text{fst } \text{lr} \cdot \sigma. u \in \text{NF-terms } Q$
 and $p2: p2 \in \text{poss } t$ and $\text{lr}: \text{lr} \in R$ and $\text{subt}: t \mid\!-\ p2 = \text{fst } \text{lr} \cdot \sigma$
 and $s: s = \text{replace-at } t \ p2 \ (\text{snd } \text{lr} \cdot \sigma)$
 and $\text{nfs}: \text{NF-subst nfs lr } \sigma \ Q$ by auto
 show ?thesis unfolding *qrstep-qrstps-r-p-s-conv*
proof (*intro exI*)
 show $(?one, ?two) \in \text{qrstep-r-p-s nfs } Q \ R \text{ lr } p2 \ \sigma$
 unfolding d
 apply (*intro CollectI, unfold split fst-conv snd-conv*)
 apply (*unfold parallel-replace-at-subt-at*[*OF p12 p1 p2*])
 apply (*intro conjI NF subt lr nfs*)
 using $s \text{ parallel-poss-replace-at}$ [*OF p12 p1*] $p2 \text{ parallel-replace-at}$ [*OF p12 p1*]
 by auto
 qed
 qed

the advantage of the following lemma is the fact, that w.l.o.g. for termination

analysis one can first reduce the argument s of the context to Q-normal form

lemma *normalize-subterm-SN*:

assumes *SN-s*: $SN\text{-on } (qrstep\ nfs\ Q\ R)\ \{s\}$
and *SN-replace*: $\bigwedge t. (s, t) \in (qrstep\ nfs\ Q\ R)^* \implies t \in NF\text{-terms } Q \implies SN\text{-on } (qrstep\ nfs\ Q\ R)\ \{C\langle t \rangle\}$
and *SN-C*: $SN\text{-on } (qrstep\ nfs\ Q\ R)\ (set\ (ctx\text{-to-term-list } C))$
shows $SN\text{-on } (qrstep\ nfs\ Q\ R)\ \{C\langle s \rangle\}$ **(is** $SN\text{-on } ?R\text{-})$

proof

fix f
assume $f\ 0 \in \{C\langle s \rangle\}$ **and** *steps*: $\forall i. (f\ i, f\ (Suc\ i)) \in ?R$
then have $zero: f\ 0 = C\langle s \rangle$ **by** *auto*
from *choice*[*OF steps*[*unfolded qrstep-qrstep-r-p-s-conv*]]
obtain lr **where** $\forall i. \exists p\ \sigma. (f\ i, f\ (Suc\ i)) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ (lr\ i)\ p\ \sigma$ **by** *auto*
from *choice*[*OF this*] **obtain** p **where** $\forall i. \exists \sigma. (f\ i, f\ (Suc\ i)) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ (lr\ i)\ (p\ i)\ \sigma$ **by** *auto*
from *choice*[*OF this*] **obtain** σ **where** *steps*: $\bigwedge i. (f\ i, f\ (Suc\ i)) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ (lr\ i)\ (p\ i)\ (\sigma\ i)$ **by** *auto*
let $?p = hole\text{-}pos\ C$
{
fix i
assume *below-or-par*: $\bigwedge j. j < i \implies ?p \leq_p p\ j \vee ?p \perp p\ j$
{
fix j
assume $j: j \leq i$
then have $?p \in poss\ (f\ j) \wedge (s, f\ j \mid -\ ?p) \in ?R^*$
proof (*induct j*)
case 0
show *?case* **unfolding** *zero* **by** *simp*
next
case $(Suc\ j)$
then have $p: ?p \in poss\ (f\ j)$ **and** *ssteps*: $(s, f\ j \mid -\ ?p) \in ?R^*$ **by** *auto*
from *Suc*(2) **have** $j < i$ **by** *auto*
from *below-or-par*[*OF this*]
show *?case*
proof
assume *True*: $?p \leq_p p\ j$
then obtain q **where** $q: ?p @\ q = p\ j$ **unfolding** *less-eq-pos-def* **by** *auto*
from *qrstep-subt-at-gen*[*OF steps*[*of j, unfolded q[symmetric]*]]
have *step*: $(f\ j \mid -\ ?p, f\ (Suc\ j) \mid -\ ?p) \in ?R$ **unfolding** *qrstep-qrstep-r-p-s-conv*
by *blast*
from *steps*[*of j, unfolded qrstep-r-p-s-def*]
have *id*: $f\ (Suc\ j) = replace\text{-}at\ (f\ j)\ (p\ j)\ (snd\ (lr\ j) \cdot \sigma\ j)$ **and** *pi*: $p\ j \in poss\ (f\ j)$ **by** *auto*
from *replace-at-below-poss*[*OF pi True*] *ssteps step*
show *?thesis* **unfolding** *id* **by** *auto*
next
assume *par*: $?p \perp p\ j$
from *parallel-qrstep-subt-at*[*OF steps*[*of j*] *parallel-pos-sym*[*OF par*] p]

```

ssteps
  show ?thesis by auto
qed
qed
}
then have ?p ∈ poss (f i) ∧ (s, f i |- ?p) ∈ ?R* by auto
} note poss-rewrite = this
show False
proof (cases ∃ i. p i <p ?p)
  case False
  {
    fix i
    from False have ¬ (p i <p ?p) by auto
    with pos-cases[of ?p p i] have ?p ≤p p i ∨ ?p ⊥ p i by auto
  } note below-or-par = this
  from poss-rewrite[OF this]
  have p: ∧ i. ?p ∈ poss (f i) ..
  from below-or-par
  have (INFM i. ?p ≤p p i) ∨ ¬ (INFM i. ?p ≤p p i) by auto
  then show False
proof
  assume inf: INFM i. ?p ≤p p i
  let ?i = λ i. ?p ≤p p i
  interpret infinitely-many ?i
  by (unfold-locales, rule inf)
  obtain g where g: g = (λ i. f i |- ?p) by auto
  {
    fix i
    from index-p[of i]
    obtain q where q: ?p @ q = p (index i) unfolding less-eq-pos-def by auto
    from qrstep-subt-at-gen[OF steps[of index i, unfolded q[symmetric]]]
    have (g (index i), g (Suc (index i))) ∈ ?R unfolding g qrstep-qrstep-r-p-s-conv
  }
by blast
} note gsteps = this
{
  fix i
  assume ¬ ?i i
  with below-or-par[of i] have par: ?p ⊥ p i by simp
  from parallel-qrstep-subt-at[OF steps[of i] parallel-pos-sym[OF par] p]
  have g i = g (Suc i) unfolding g by simp
} note gid = this
obtain h where h: h = (λ i. g (index i)) by auto
{
  fix i
  from index-ordered[of i] have Suc (index i) ≤ index (Suc i) by simp
  then have ∃ j. index (Suc i) = Suc (index i) + j by arith
  then obtain j where id: index (Suc i) = Suc (index i) + j by auto
  {
    fix j

```

```

    assume  $Suc\ (index\ i) + j \leq index\ (Suc\ i)$ 
    then have  $g\ (Suc\ (index\ i)) = g\ (Suc\ (index\ i) + j)$ 
    proof (induct j)
      case 0 show ?case by simp
    next
      case (Suc j)
      then have  $g\ (Suc\ (index\ i)) = g\ (Suc\ (index\ i) + j)$  by simp
      also have  $\dots = g\ (Suc\ (Suc\ (index\ i) + j))$ 
      by (rule gid[OF index-not-p-between, of i], insert Suc(2), auto)
      finally show ?case by simp
    qed
  }
  from this[of j] have  $g\ (Suc\ (index\ i)) = h\ (Suc\ i)$  unfolding h id by simp
  with gsteps[of i] have  $(h\ i, h\ (Suc\ i)) \in ?R$  unfolding h by simp
}
then have  $\neg SN\text{-on}\ ?R\ \{h\ 0\}$  by auto
then have  $nSN: \neg SN\text{-on}\ ?R\ \{g\ (index\ 0)\}$  unfolding h .
obtain iz where iz:  $iz = index\ 0$  by auto
from gid[OF index-not-p-start] have  $\bigwedge i. i < iz \implies g\ i = g\ (Suc\ i)$  unfolding
iz by auto
then have  $g\ 0 = g\ iz$ 
  by (induct iz, auto)
with nSN have  $nSN: \neg SN\text{-on}\ ?R\ \{g\ 0\}$  unfolding iz by simp
with SN-s show False unfolding g zero by simp
next
  assume inf:  $\neg (INFM\ i. ?p \leq_p p\ i)$ 
  let ?cc =  $\lambda i. ctxt\text{-to-term-list}\ (ctxt\text{-of-pos-term}\ ?p\ (f\ i))$ 
  let ?step =  $\lambda i\ j\ n. j < n \wedge (?cc\ i\ !\ j, ?cc\ (Suc\ i)\ !\ j) \in ?R \wedge (\forall k. k \neq j \longrightarrow ?cc\ (Suc\ i)\ !\ k = ?cc\ i\ !\ k) \wedge length\ (?cc\ (Suc\ i)) = n$ 
  {
    fix i
    assume ?p  $\perp$  p i
    from parallel-grstep-ctxt-to-term-list[OF this p steps]
    obtain p' j where p':  $p' \leq_p p\ i$  and j:  $j < length\ (?cc\ i)$ 
      and i:  $?cc\ i\ !\ j = f\ i\ \mid\text{-}\ p'$ 
      and si:  $?cc\ (Suc\ i) = (?cc\ i)\ [j := f\ (Suc\ i)\ \mid\text{-}\ p']$  by blast+
    from si have len:  $length\ (?cc\ (Suc\ i)) = length\ (?cc\ i)$  by simp
    from p' obtain q where p':  $p' @ q = p\ i$  unfolding less-eq-pos-def by
blast
    then have pi:  $p\ i = p' @ q$  by simp
    from grstep-subt-at-gen[OF steps[of i, unfolded pi]]
    have  $(?cc\ i\ !\ j, ?cc\ (Suc\ i)\ !\ j) \in grstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ (lr\ i)\ q\ (\sigma\ i)$  unfolding
i si using j
      by auto
    then have step:  $(?cc\ i\ !\ j, ?cc\ (Suc\ i)\ !\ j) \in ?R$  unfolding grstep-grstep-r-p-s-conv
by blast
    have id:  $\forall k. k \neq j \longrightarrow ?cc\ (Suc\ i)\ !\ k = ?cc\ i\ !\ k$  unfolding si by auto
    from step id j have  $\exists j. ?step\ i\ j\ (length\ (?cc\ i))$ 
      unfolding len by blast

```

```

} note par-step = this
{
  fix i
  assume  $\neg (?p \perp p \ i)$ 
  with below-or-par have  $?p \leq_p p \ i$  by auto
  from ctxt-of-pos-term-grstep-below[OF steps this]
  have  $?cc \ (Suc \ i) = ?cc \ i$  by simp
} note below-step = this
obtain n where n:  $n = length \ (?cc \ 0)$  by simp
{
  fix i
  have  $length \ (?cc \ i) = n$ 
  proof (induct i)
    case 0 show ?case unfolding n by simp
  next
    case (Suc i)
    then show ?case
      using below-step[of i] par-step[of i]
      by (cases  $?p \perp p \ i$ , auto)
  qed
} note len = this
from par-step have par-step:  $\bigwedge i. ?p \perp p \ i \implies (\exists j. ?step \ i \ j \ n)$  unfolding
len by simp
from inf obtain k where par:  $\bigwedge j. j \geq k \implies \neg ?p \leq_p p \ j$  unfolding
INFM-nat-le by auto
with below-or-par have par:  $\bigwedge j. j \geq k \implies ?p \perp p \ j$  by auto
let  $?jstep = \lambda i \ j. (?cc \ (i + k) \ ! \ j, ?cc \ (Suc \ i + k) \ ! \ j) \in ?R$ 
{
  fix i
  have  $i + k \geq k$  by simp
  from par-step[OF par[OF this]]
  obtain j where j:  $j < n$  and step:  $?jstep \ i \ j$  by auto
  then have  $\exists j < n. ?jstep \ i \ j$  by blast
}
then have  $\forall i. \exists j < n. ?jstep \ i \ j$  by blast
from inf-pigeonhole-principle[OF this] obtain j where j:  $j < n$  and steps:
 $\bigwedge i. \exists i' \geq i. ?jstep \ i' \ j$  by blast
let  $?t = \lambda i. ?cc \ i \ ! \ j$ 
{
  fix i
  have  $(?t \ i, ?t \ (Suc \ i)) \in ?R^{\hat{=}}$ 
  proof (cases  $?p \perp p \ i$ )
    case False
    from below-step[OF this] show ?thesis by simp
  next
    case True
    from par-step[OF this]
    obtain k where  $?step \ i \ k \ n$  by auto
    then show ?thesis by (cases  $k = j$ , auto)
  
```

```

    qed
  } then have rsteps:  $\forall i. (?t\ i, ?t\ (Suc\ i)) \in Id \cup ?R$  by blast
  from  $j[unfolded\ len[of\ 0, symmetric]]$  have  $t0: ?t\ 0 \in set\ (ctxt\text{-}to\text{-}term\text{-}list$ 
C)
    unfolding zero set-conv-nth by auto
  with SN-C have SN: SN-on  $?R\ \{?t\ 0\}$  unfolding SN-on-def by simp
  from non-strict-ending[OF rsteps] SN
  obtain  $k''$  where  $\forall k' \geq k''. (?t\ k', ?t\ (Suc\ k')) \notin ?R$  by blast
  with steps[of  $k''$ ] show False by auto
qed
next
case True
let  $?P = \lambda i. p\ i \leq_p\ ?p$ 
from LeastI-ex[of  $?P, OF\ True$ ] have  $?P\ (LEAST\ i. ?P\ i)$  .
then obtain  $i$  where  $i: i = (LEAST\ i. ?P\ i)$  and  $pi: ?P\ i$  by auto
{
  fix  $j$ 
  assume  $j: j < i$ 
  from not-less-Least[OF this[unfolded  $i$ ]] have  $\neg ?P\ j$  .
  with pos-cases[of  $?p\ p\ j$ ] have  $?p \leq_p\ p\ j \vee ?p \perp p\ j$  by auto
} note below-or-par = this
from poss-rewrite[OF this]
have  $p: ?p \in poss\ (f\ i)$  and rewr:  $(s, f\ i \mid -\ ?p) \in ?R^*$  by auto
from steps[of  $i, unfolded\ qrstep\text{-}r\text{-}p\text{-}s\text{-}def$ ] have NF:  $\bigwedge u. u \triangleleft f\ i \mid -\ p\ i \implies u$ 
 $\in NF\text{-}terms\ Q$  by auto
from NF[OF less-pos-imp-supt[OF  $pi\ p$ ]] have NF:  $f\ i \mid -\ ?p \in NF\text{-}terms\ Q$  .
from SN-replace[OF rewr NF] have SN: SN-on  $?R\ \{C\langle f\ i \mid -\ ?p \rangle\}$  .
{
  fix  $w$ 
  from below-or-par have  $(C\langle w \rangle, replace\text{-}at\ (f\ i)\ ?p\ w) \in ?R^*$ 
  proof (induct  $i$ )
    case 0
    show ?case unfolding zero by simp
  next
  case (Suc  $i$ )
  then have steps':  $(C\langle w \rangle, replace\text{-}at\ (f\ i)\ ?p\ w) \in ?R^*$  by auto
  from poss-rewrite[OF Suc(2)] have  $p: ?p \in poss\ (f\ i)$  by auto
  from Suc(2)[of  $i$ ] have  $?p \leq_p\ p\ i \vee ?p \perp p\ i$  by auto
  then have  $(replace\text{-}at\ (f\ i)\ ?p\ w, replace\text{-}at\ (f\ (Suc\ i))\ ?p\ w) \in ?R^*$ 
  proof
    assume  $?p \perp p\ i$ 
    from parallel-qrstep-r-p-s[OF this  $p\ steps[of\ i]$ ]
    show ?thesis by auto
  next
  assume  $?p \leq_p\ p\ i$ 
  from ctxt-of-pos-term-replace-at-below[OF  $p\ this$ ]
  have  $replace\text{-}at\ (f\ (Suc\ i))\ ?p\ w = replace\text{-}at\ (f\ i)\ ?p\ w$ 
  using steps[of  $i, unfolded\ qrstep\text{-}r\text{-}p\text{-}s\text{-}def$ ] by auto
  then show ?thesis by auto

```

```

      qed
      with steps' show ?case by auto
    qed
  }
  from this[of f i |- ?p] have steps': (C⟨f i |- ?p⟩, f i) ∈ ?R*
    unfolding ctxt-supt-id[OF p] .
  obtain g where g: g = (λ j. f (j + i)) by auto
  from steps-preserve-SN-on[OF steps' SN] have SN: SN-on ?R {g 0} unfolding
g by auto
  {
    fix j
    from steps[of j + i] have (f (j + i), f (Suc (j + i))) ∈ ?R
      unfolding grstep-grstep-r-p-s-conv by blast
    then have (g j, g (Suc j)) ∈ ?R unfolding g by auto
  }
  then have ¬ SN-on ?R {g 0} by auto
  with SN show False by simp
qed
qed

```

lemma *NF-rstep-supt-args-conv*:

$(\forall u \triangleleft t. u \in NF \text{ (rstep } R)) = (\forall u \in \text{set (args } t). u \in NF \text{ (rstep } R))$

proof (cases t)

case (Var x) show ?thesis unfolding Var by auto

next

case (Fun f ts)

show ?thesis (is ?lhs = ?rhs)

proof

assume ?lhs then show ?rhs by (auto simp: Fun)

next

assume ?rhs show ?lhs

proof (intro allI impI)

fix u assume u < t

from supt-Fun-imp-arg-supteq[OF this[unfolded Fun]]

obtain t where t ∈ set ts and t ≥ u by best

from ⟨?rhs⟩[unfolded Fun] and ⟨t ∈ set ts⟩ have t ∈ NF (rstep R) by simp

from this and ⟨t ≥ u⟩ show u ∈ NF (rstep R) by (rule NF-subterm)

qed

qed

qed

lemma *NF-terms-args-conv*:

$(\forall u \in \text{set (args } t). u \in NF\text{-terms } T) = (\forall u \triangleleft t. u \in NF\text{-terms } T)$

using NF-rstep-supt-args-conv[symmetric, of t Id-on T] .

lemma *ctxt-closed-grstep* [intro]: *ctxt.closed* (*grstep nfs Q R*)

unfolding *ctxt.closed-def*

proof (rule subrelI)

fix s t

assume $(s, t) \in \text{ctxt.closure } (\text{qrstep nfs } Q \ R)$
then show $(s, t) \in \text{qrstep nfs } Q \ R$ **by** *induct auto*
qed

lemma *qrsteps-ctxt-closed*:
assumes $(s, t) \in (\text{qrstep nfs } Q \ R)^*$
shows $(C\langle s \rangle, C\langle t \rangle) \in (\text{qrstep nfs } Q \ R)^*$
by (*rule ctxt.closedD[OF - assms], blast*)

lemma *not-NF-rstep-minimal*:
assumes $s \notin \text{NF } (\text{rstep } R)$
shows $\exists t \triangleleft s. t \notin \text{NF } (\text{rstep } R) \wedge (\forall u \triangleleft t. u \in \text{NF } (\text{rstep } R))$
using *assms* **proof** (*induct s rule: subterm-induct*)
case (*subterm v*)
then show ?*case*
proof (*cases* $\forall w \triangleleft v. w \in \text{NF } (\text{rstep } R)$)
case *True* **with** *subterm(2)* **show** ?*thesis* **by** *auto*
next
case *False*
then obtain *w* **where** $v \triangleright w$ **and** $w \notin \text{NF } (\text{rstep } R)$ **by** *auto*
with *subterm(1)* **obtain** *t* **where** $w \triangleright t$ **and** *notNF*: $t \notin \text{NF } (\text{rstep } R)$
and *NF*: $\forall u \triangleleft t. u \in \text{NF } (\text{rstep } R)$ **by** *auto*
from $\langle v \triangleright w \rangle$ **and** $\langle w \triangleright t \rangle$ **have** $v \triangleright t$ **using** *supt-supteq-trans[of v w t]* **by**
auto
with *notNF* **and** *NF* **show** ?*thesis* **by** *auto*
qed
qed

lemma *Var-NF-terms*: **assumes** *no-lhs-var*: $\bigwedge l. l \in Q \implies \text{is-Fun } l$
shows $\text{Var } x \in \text{NF-terms } Q$
proof (*rule ccontr*)
assume $\text{Var } x \notin \text{NF-terms } Q$
then obtain $l \ C \ \sigma$ **where** $l: l \in Q$ **and** $x: \text{Var } x = C \langle l \cdot \sigma \rangle$ **by** *blast*
from *x* **have** $\text{Var } x = l \cdot \sigma$ **by** (*cases C, auto*)
with *no-lhs-var*[*OF l*] **show** *False* **by** *auto*
qed

lemma *rstep-imp-irstep*:
assumes *st*: $(s, t) \in \text{rstep } R$ **and** *no-lhs-var*: $\bigwedge l \ r. \text{nfs} \implies (l, r) \in R \implies \text{is-Fun } l$
shows $\exists u. (s, u) \in \text{irstep nfs } R$
proof –
from *assms* **have** $s \notin \text{NF } (\text{rstep } R)$ **by** *auto*
from *not-NF-rstep-minimal*[*OF this*] **obtain** *u* **where** $s \triangleright u$ **and** *notNF*: $u \notin \text{NF } (\text{rstep } R)$
and *NF*: $\forall w \triangleleft u. w \in \text{NF } (\text{rstep } R)$ **by** *auto*
from *notNF* **obtain** *v* **where** $(u, v) \in \text{rstep } R$ **by** *auto*
then obtain $l \ r \ C \ \sigma$ **where** $(l, r) \in R$ **and** $u: u = C\langle l \cdot \sigma \rangle$ **and** $v: v = C\langle r \cdot$

```

 $\sigma\rangle$  by auto
from u have  $u \supseteq l \cdot \sigma$  by auto
have nf:  $\forall w \triangleleft l \cdot \sigma. w \in NF \text{ (rstep } R)$ 
proof (intro allI impI)
  fix w assume  $l \cdot \sigma \triangleright w$ 
  with  $\langle u \supseteq l \cdot \sigma \rangle$  have  $u \triangleright w$  by (rule supseq-supt-trans)
  with NF show  $w \in NF \text{ (rstep } R)$  by auto
qed
let ?sigma =  $\lambda x. \text{ if } x \in \text{vars-term } l \text{ then } \sigma \ x \text{ else } \text{Var } x$ 
obtain v where  $v = C \langle r \cdot ?sigma \rangle$  by auto
have lsig:  $l \cdot \sigma = l \cdot ?sigma$ 
  by (rule term-subst-eq, auto)
have NF-subst nfs (l,r) ?sigma (lhss R)
  unfolding NF-subst-def
proof (intro impI subsetI)
  fix t
  assume nfs: nfs
  assume t  $\in$  ?sigma 'vars-rule (l,r)'
  then obtain x where  $x: x \in \text{vars-rule } (l,r)$  and t:  $t = ?sigma \ x$  by auto
  show  $t \in NF\text{-terms (lhss } R)$ 
  proof (cases  $x \in \text{vars-term } l$ )
    case True
    then have  $\text{Var } x \trianglelefteq l$  by auto
    with no-lhs-var[OF nfs  $\langle (l,r) \in R \rangle$ ] have  $\text{Var } x \triangleleft l$  by (cases l, auto)
    then have  $\text{Var } x \cdot \sigma \triangleleft l \cdot \sigma$  by (rule supt-subst)
    with nf show ?thesis unfolding t using True by simp
  next
    case False
    then have t:  $t = \text{Var } x$  unfolding t by auto
    show ?thesis unfolding t
      by (rule Var-NF-terms, insert no-lhs-var[OF nfs], auto)
  qed
qed
with nf  $\langle (l, r) \in R \rangle$  and u and v have  $(u, v) \in \text{irstep nfs } R$ 
  unfolding irstep-def and NF-terms-lhss[symmetric, of R] lsig by blast
from  $\langle s \supseteq u \rangle$  obtain C where  $s = C\langle u \rangle$  by auto
from  $\langle (u, v) \in \text{irstep nfs } R \rangle$  have  $(C\langle u \rangle, C\langle v \rangle) \in \text{irstep nfs } R$  unfolding
  irstep-def by auto
  then show ?thesis unfolding  $\langle s = C\langle u \rangle \rangle$  by best
qed

lemma NF-irstep-NF-rstep:
  assumes no-lhs-var:  $\bigwedge l \ r. \text{ nfs} \implies (l,r) \in R \implies \text{is-Fun } l$ 
  shows  $NF \text{ (irstep nfs } R) = NF \text{ (rstep } R)$ 
proof -
  have  $\text{irstep nfs } R \subseteq \text{rstep } R$  unfolding irstep-def ..
  from NF-anti-mono[OF this] have  $NF \text{ (rstep } R) \subseteq NF \text{ (irstep nfs } R)$  .
  moreover have  $NF \text{ (irstep nfs } R) \subseteq NF \text{ (rstep } R)$ 
  proof

```

```

fix  $s$  assume  $s \in NF$  (irstep nfs R) show  $s \in NF$  (rstep R)
proof (rule ccontr)
  assume  $s \notin NF$  (rstep R)
  then obtain  $t$  where  $(s, t) \in rstep R$  by auto
  from rstep-imp-irstep[OF this no-lhs-var, of nfs]
    obtain  $u$  where  $(s, u) \in irstep nfs R$  by force
  with  $\langle s \in NF$  (irstep nfs R) show False by auto
qed
qed
ultimately show ?thesis by simp
qed

```

definition

```

applicable-rule :: (f, v) terms  $\Rightarrow$  (f, v) rule  $\Rightarrow$  bool
where
  applicable-rule  $Q$   $lr \equiv \forall s \triangleleft fst\ lr. s \in NF\text{-terms } Q$ 

```

lemma *applicable-rule-empty*: *applicable-rule* {} lr
unfolding *applicable-rule-def* **by** *auto*

lemma *only-applicable-rules*:

```

assumes  $\forall u \triangleleft l \cdot \sigma. u \in NF\text{-terms } Q$ 
shows applicable-rule  $Q$  ( $l, r$ )
unfolding applicable-rule-def
proof (intro allI impI)
  fix  $s$  assume  $fst\ (l, r) \triangleright s$ 
  then have  $l \triangleright s$  by simp
  then have  $l \cdot \sigma \triangleright s \cdot \sigma$  by (rule supt-subst)
  with assms have  $s \cdot \sigma \in NF\text{-terms } Q$  by simp
  from NF-instance[OF this] show  $s \in NF\text{-terms } Q$  .
qed

```

definition

```

applicable-rules :: (f, v) terms  $\Rightarrow$  (f, v) trs  $\Rightarrow$  (f, v) trs
where
  applicable-rules  $Q$   $R \equiv \{(l, r) \mid l\ r. (l, r) \in R \wedge \text{applicable-rule } Q\ (l, r)\}$ 

```

lemma *applicable-rules-union*: *applicable-rules* Q ($R \cup S$) = *applicable-rules* Q R
 \cup *applicable-rules* Q S
unfolding *applicable-rules-def* **by** *auto*

lemma *applicable-rules-empty*[*simp*]:

```

applicable-rules {}  $R = R$ 
unfolding applicable-rules-def using applicable-rule-empty by auto

```

lemma *applicable-rules-subset*: *applicable-rules* Q $R \subseteq R$
unfolding *applicable-rules-def* *applicable-rule-def* **by** *auto*

lemma *qrstep-applicable-rules*: $qrstep\ nfs\ Q\ (applicable\text{-}rules\ Q\ R) = qrstep\ nfs\ Q\ R$

proof –

let $?U = applicable\text{-}rules\ Q\ R$

show *?thesis*

proof

show $qrstep\ nfs\ Q\ ?U \subseteq qrstep\ nfs\ Q\ R$ **by** (rule *qrstep-rules-mono*[*OF applicable-rules-subset*])

next

show $qrstep\ nfs\ Q\ R \subseteq qrstep\ nfs\ Q\ ?U$

proof (rule *subrelI*)

fix $s\ t$

assume $(s, t) \in qrstep\ nfs\ Q\ R$

then obtain $l\ r\ C\ \sigma$ **where** $(l, r) \in R$ **and** $NF: \forall u \triangleleft l \cdot \sigma. u \in NF\text{-}terms\ Q$

and $s = C\langle l \cdot \sigma \rangle$ **and** $t = C\langle r \cdot \sigma \rangle$

and $nfs: NF\text{-}subst\ nfs\ (l, r)\ \sigma\ Q$ **by** *auto*

with *only-applicable-rules*[*OF NF*] **have** $(l, r) \in ?U$ **by** (*auto simp: applicable-rules-def*)

with NF **and** $\langle (l, r) \in R \rangle$ **and** s **and** t **and** nfs **show** $(s, t) \in qrstep\ nfs\ Q$

?U by *auto*

qed

qed

qed

lemma *ndef-applicable-rules*: $\neg defined\ R\ x \implies \neg defined\ (applicable\text{-}rules\ Q\ R)\ x$

unfolding *defined-def applicable-rules-def* **by** *auto*

lemma *nvar-qrstep-Fun*:

assumes $nvar: \forall (l, r) \in R. is\text{-}Fun\ l$

and $step: (s, t) \in qrstep\ nfs\ Q\ R$

shows $\exists f\ ts. s = Fun\ f\ ts$

proof (*cases s*)

case ($Var\ x$)

from $step$ **obtain** $C\ \sigma\ l\ r$ **where** $x: Var\ x = C\langle l \cdot \sigma \rangle$ **and** $lr: (l, r) \in R$

unfolding Var

by *auto*

from $nvar\ lr$ **obtain** $f\ ls$ **where** $l: l = Fun\ f\ ls$ **by** (*cases l, auto*)

then show *?thesis* **using** x **by** (*cases C, auto*)

qed *auto*

weakly well-formedness (all applicable rules are well-formed).

definition

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

where

$wf\text{-}rule\ Q\ lr \equiv$

$applicable\text{-}rule\ Q\ lr \longrightarrow (is\text{-}Fun\ (fst\ lr) \wedge vars\text{-}term\ (snd\ lr) \subseteq vars\text{-}term\ (fst\ lr))$

definition

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

where

$wf-qtrs \ Q \ R \equiv \forall (l, r) \in R. \text{applicable-rule } Q \ (l, r) \longrightarrow (\text{is-Fun } l \wedge \text{vars-term } r \subseteq \text{vars-term } l)$

lemma $wf-qtrs-wf-rules$: $wf-qtrs \ Q \ R = (\forall (l, r) \in R. \text{wf-rule } Q \ (l, r))$

unfolding $wf-qtrs-def \ wf-rule-def$ **by** *auto*

lemma $wf-qtrs-wf-trs$: $wf-qtrs \ Q \ R = \text{wf-trs } (\text{applicable-rules } Q \ R)$

unfolding $wf-qtrs-def \ wf-trs-def \ applicable-rules-def$ **by** *force*

lemma $wf-qtrs-empty$: $wf-qtrs \ \{\} \ R = \text{wf-trs } R$

unfolding $wf-qtrs-wf-trs \ applicable-rules-empty$ **by** *simp*

lemma $left\text{-Var-imp-not-SN-grstep}$:

assumes xr : $(\text{Var } x, r) \in R$ **and** nfs : $\neg \text{nfs}$ **shows** $\neg (\text{SN-on } (\text{grstep } nfs \ Q \ R) \ \{t\})$

proof –

have $steps$: $\forall t. \exists s. (t, s) \in \text{grstep } nfs \ Q \ R$

proof

fix t

show $\exists s. (t, s) \in \text{grstep } nfs \ Q \ R$

proof (*induct* t)

case $(\text{Var } y)$

let $?xt = \text{subst } x \ (\text{Var } y)$

let $?rxt = r \cdot ?xt$

have NF : $\forall u \triangleleft \text{Var } x \cdot ?xt. u \in \text{NF-terms } Q$ **by** *auto*

have $(\text{Var } x \cdot ?xt, ?rxt) \in \text{grstep } nfs \ Q \ R$

by (*rule* $\text{grstep.subst}[OF \ NF \ xr, \text{of } nfs]$, *insert* nfs , *simp*)

then show $?case$ **by** *auto*

next

case $(\text{Fun } f \ ss)$

show $?case$

proof (*cases* ss)

case Nil

let $?xt = \text{subst } x \ (\text{Fun } f \ ss)$

let $?rxt = r \cdot ?xt$

have $\forall u \triangleleft \text{Var } x \cdot ?xt. u \in \text{NF-terms } Q$ **unfolding** Nil **by** *auto*

from $\text{grstep.subst}[OF \ \text{this } xr, \text{of } nfs]$

have $(\text{Var } x \cdot ?xt, ?rxt) \in \text{grstep } nfs \ Q \ R$ **using** nfs **by** *simp*

then show $?thesis$ **by** *auto*

next

case $(\text{Cons } s \ ts)$

with Fun **obtain** t **where** step : $(s, t) \in \text{grstep } nfs \ Q \ R$ **by** *auto*

let $?C = \text{More } f \ \square \ \square \ ts$

from Cons **have** id : $\text{Fun } f \ ss = ?C \langle s \rangle$ **by** *auto*

have $(?C \langle s \rangle, ?C \langle t \rangle) \in \text{ctxt.closure } (\text{grstep } nfs \ Q \ R)$ **by** (*blast intro*: *step*)

with $\text{ctxt-closed-grstep}[of \ nfs \ Q \ R]$

```

    have (Fun f ss, ?C⟨t⟩) ∈ qrstep nfs Q R
      unfolding ctxt.closed-def id by auto
    then show ?thesis by auto
  qed
qed
qed
from choice[OF this]
obtain f where steps:  $\bigwedge t. (t, f t) \in qrstep\ nfs\ Q\ R$  by blast
show ?thesis
  by (rule steps-imp-not-SN-on, rule steps)
qed

lemma ctxt-compose-Suc:  $(C \hat{\ } Suc\ i)\langle t \rangle = (C \hat{\ } i)\langle C\langle t \rangle \rangle$ 
using ctxt-power-compose-distr[of C i Suc 0] by simp

lemma SN-on-imp-wuf-rule:
  assumes SN: SN-on (qrstep nfs Q R) {t}
    and t:  $t = C\langle l \cdot \sigma \rangle$  and lr:  $(l, r) \in R$  and NF:  $\forall u \triangleleft l \cdot \sigma. u \in NF\text{-terms}\ Q$ 
    and nfs:  $\neg\ nfs$ 
  shows wuf-rule Q (l, r)
proof (cases vars-term r  $\subseteq$  vars-term l)
case True
  from left-Var-imp-not-SN-qrstep[of - - R nfs Q t] lr SN nfs have is-Fun l by
(induct l) auto
  with True show ?thesis unfolding wuf-rule-def by auto
next
case False
  then obtain x where  $x \in vars\text{-term}\ r - vars\text{-term}\ l$  by auto
  then have not-in-l:  $x \notin vars\text{-term}\ l$  by auto
  from only-applicable-rules[OF NF] have applicable-rule Q (l,r) .
  let ? $\delta$  = ( $\lambda y. \text{if } y = x \text{ then } l \cdot \sigma \text{ else } \sigma\ y$ )
  have  $l \cdot \sigma = l \cdot (\sigma \mid s\ (vars\text{-term}\ l))$  by (rule coincidence-lemma)
  also have  $\dots = l \cdot (? \delta \mid s\ (vars\text{-term}\ l))$ 
  proof (rule arg-cong[of - - ( $\cdot$ ) l])
    show  $\sigma \mid s\ vars\text{-term}\ l = ? \delta \mid s\ vars\text{-term}\ l$ 
  proof -
    have main: (if  $y \in vars\text{-term}\ l$  then  $\sigma\ y$  else Var y) =
      (if  $y \in vars\text{-term}\ l$  then if  $y = x$  then  $l \cdot \sigma$  else  $\sigma\ y$  else Var y) for y
    by (cases  $y = x$ , auto simp: not-in-l)
    show ?thesis unfolding subst-restrict-def by (rule ext, simp add: main)
  qed
  qed
qed
also have  $\dots = l \cdot ? \delta$  by (rule coincidence-lemma[symmetric])
finally have lsld:  $l \cdot \sigma = l \cdot ? \delta$  .
from  $\langle x \in vars\text{-term}\ r - vars\text{-term}\ l \rangle$  have in-r:  $x \in vars\text{-term}\ r$  by auto
from supseq-Var[OF in-r] obtain C where  $r = C\langle Var\ x \rangle$  by auto
then have  $r \cdot ? \delta = (C \cdot_c ? \delta)\langle l \cdot ? \delta \rangle$  by (auto simp: lsld)
then obtain C where rd:  $r \cdot ? \delta = C\langle l \cdot ? \delta \rangle$  by auto
obtain ls where  $ls: l \cdot \sigma = ls$  by auto

```

obtain f **where** $f: \bigwedge i. f\ i = (C \hat{\ } i) \langle l \cdot ?\delta \rangle$ **by** *auto*
have *steps*: *chain* (*qrstep* *nfs* $Q\ R$) f
proof
 fix i
 show $(f\ i, f\ (Suc\ i)) \in qrstep\ nfs\ Q\ R$
 proof
 show $\forall u \triangleleft l \cdot ?\delta. u \in NF\text{-terms}\ Q$ **unfolding** *lsld*[*symmetric*] **by** *fact*
 show $(l, r) \in R$ **by** *fact*
 show $f\ i = (C \hat{\ } i) \langle l \cdot ?\delta \rangle$ **by** *fact*
 show $f\ (Suc\ i) = (C \hat{\ } i) \langle r \cdot ?\delta \rangle$ **using** $f[of\ Suc\ i, unfolded\ ctxt\text{-compose}\text{-}Suc\ rd[*symmetric*]]$.
 qed (*insert* *nfs*, *auto*)
 qed
 have *start*: $f\ 0 = l \cdot \sigma$
 by (*simp* *add*: $f\ lsld[*symmetric*]\ ls$)
 from *start* *steps* **have** $nSN: \neg (SN\text{-on}\ (qrstep\ nfs\ Q\ R)\ \{l \cdot \sigma\})$ **unfolding**
SN-on-def **by** *auto*
 from *not-SN-subt-imp-not-SN*[*OF* *ctxt-closed-qrstep* $nSN\ ctxt\text{-imp-supteq}$] $SN[simplified\ t]$
 have *False* ..
 then **show** *?thesis* ..
qed

lemma *SN-imp-wwf-qtrs*:

assumes $SN\ (qrstep\ nfs\ Q\ R)$ **and** $nfs: \neg nfs$ **shows** $wwf\text{-}qtrs\ Q\ R$
proof (*rule* *ccontr*)
 assume $\neg wwf\text{-}qtrs\ Q\ R$
 then **obtain** $l\ r$ **where** $lr: (l, r) \in R$ **and** $not\text{-}wwf: \neg wwf\text{-}rule\ Q\ (l, r)$
 unfolding *wwf-qtrs-wwf-rules* **by** *auto*
 then **have** *applicable*: *applicable-rule* $Q\ (l, r)$ **unfolding** *wwf-rule-def* **by** *auto*
 from *assms* **have** $SN: SN\text{-on}\ (qrstep\ nfs\ Q\ R)\ \{l\}$ **unfolding** *SN-defs* **by** *auto*
 from *SN-on-imp-wwf-rule*[*OF* $SN - lr - nfs, of\ \square\ Var$] **and** *not-wwf*
 have $\neg (\forall u \triangleleft l. u \in NF\text{-terms}\ Q)$ **by** *auto*
 then **obtain** u **where** $l \triangleright u$ **and** $not\text{-}NF: u \notin NF\text{-terms}\ Q$ **by** *auto*
 from *not-NF* **obtain** v **where** $(u, v) \in rstep\ (Id\text{-on}\ Q)$ **by** *auto*
 from $\langle l \triangleright u \rangle$ **obtain** D **where** $D \neq \square$ **and** $l = D \langle u \rangle$ **by** *auto*
 with *applicable*[*unfolded* *applicable-rule-def*] **and** $\langle (u, v) \in rstep\ (Id\text{-on}\ Q) \rangle$ **show**
False **by** *auto*
qed

lemma *left-Var-applicable*: *applicable-rule* $Q\ (Var\ x, r)$

unfolding *applicable-rule-def* **using** *Var-supt*[*of* x] **by** *auto*

lemma *wwf-qtrs-imp-left-fun*:

assumes $wwf\text{-}qtrs\ Q\ R$ **and** $(l, r) \in R$ **shows** $\exists f\ ls. l = Fun\ f\ ls$
using *assms* **unfolding** *wwf-qtrs-def*
using *left-Var-applicable*[*of* $Q - r$] **by** (*cases* l) *auto*

lemma *wwf-var-cond*:

assumes *wwf*: *wwf-qtrs* *Q R* **shows** $\forall (l, r) \in R. \text{is-Fun } l$
using *wwf-qtrs-imp-left-fun*[*OF wwf*] **by** *auto*

definition

rqrstep :: *bool* \Rightarrow (*'f*, *'v*) *terms* \Rightarrow (*'f*, *'v*) *trs* \Rightarrow (*'f*, *'v*) *term rel*

where

rqrstep nfs Q R =
 $\{(s, t). \exists l \ r \ \sigma. (\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q) \wedge (l, r) \in R \wedge s = l \cdot \sigma \wedge t = r \cdot \sigma \wedge \text{NF-subst nfs } (l, r) \ \sigma \ Q\}$

lemma *rqrstep-union*: *rqrstep nfs Q (R \cup S)* = *rqrstep nfs Q R \cup rqrstep nfs Q S*
unfolding *rqrstep-def* **by** *blast*

lemma *rqrstep-rrstep-conv*[*simp*]:

rqrstep nfs {} R = *rrstep R*
by (*auto simp: rrstep-def' rqrstep-def*)

definition

nrqrstep :: *bool* \Rightarrow (*'f*, *'v*) *terms* \Rightarrow (*'f*, *'v*) *trs* \Rightarrow (*'f*, *'v*) *term rel*

where

nrqrstep nfs Q R =
 $\{(s, t). \exists l \ r \ C \ \sigma. (\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q) \wedge (l, r) \in R \wedge C \neq \square \wedge s = C \langle l \cdot \sigma \rangle \wedge t = C \langle r \cdot \sigma \rangle \wedge \text{NF-subst nfs } (l, r) \ \sigma \ Q\}$

lemma *rqrstepI*[*intro*]:

assumes $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$ **and** $(l, r) \in R$
and $s = l \cdot \sigma$ **and** $t = r \cdot \sigma$
and *NF-subst nfs* $(l, r) \ \sigma \ Q$
shows $(s, t) \in \text{rqrstep nfs } Q \ R$
using *assms* **unfolding** *rqrstep-def* **by** *auto*

lemma *nrqrstepI*[*intro*]:

assumes $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$ **and** $(l, r) \in R$
and $C \neq \square$ **and** $s = C \langle l \cdot \sigma \rangle$ **and** $t = C \langle r \cdot \sigma \rangle$
and *NF-subst nfs* $(l, r) \ \sigma \ Q$
shows $(s, t) \in \text{nrqrstep nfs } Q \ R$
using *assms* **unfolding** *nrqrstep-def* **by** *auto*

lemma *rqrstepE*[*elim*]:

assumes $(s, t) \in \text{rqrstep nfs } Q \ R$
and $\bigwedge l \ r \ \sigma. \llbracket \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q; (l, r) \in R; s = l \cdot \sigma; t = r \cdot \sigma; \text{NF-subst nfs } (l, r) \ \sigma \ Q \rrbracket \Longrightarrow P$
shows *P*
using *assms* **unfolding** *rqrstep-def* **by** *auto*

lemma *nrqrstepE*[*elim*]:

assumes $(s, t) \in \text{nrqrstep nfs } Q \ R$

and $\bigwedge l \ r \ C \ \sigma.$
 $\llbracket \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q; (l, r) \in R; C \neq \square; s = C\langle l \cdot \sigma \rangle; t = C\langle r \cdot \sigma \rangle; \text{NF-subst nfs } (l, r) \ \sigma \ Q \rrbracket \implies P$
shows P
using *assms* **unfolding** *nrqrstep-def* **by** *auto*

lemma *rqrstep-all-mono*:

assumes $R: R \subseteq R'$ **and** $Q: \text{NF-terms } Q \subseteq \text{NF-terms } Q'$ **and** $n: Q \neq \{\}$ \implies
 $\text{nfs}' \implies \text{nfs}$

shows $\text{rqrstep nfs } Q \ R \subseteq \text{rqrstep nfs}' \ Q' \ R'$

proof

fix $s \ t$

assume $(s, t) \in \text{rqrstep nfs } Q \ R$

then obtain $l \ r \ \sigma$ **where** $s: s = l \cdot \sigma$ **and** $t: t = r \cdot \sigma$ **and** $lr: (l, r) \in R$

and $\text{NF}: \bigwedge u. u \triangleleft l \cdot \sigma \implies u \in \text{NF-terms } Q$

and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$

by *auto*

from $\text{nfs } n \ Q$ **have** $\text{nfs}: \text{NF-subst nfs}' (l, r) \ \sigma \ Q'$ **unfolding** *NF-subst-def* **by** *auto*

from $\text{NF } Q$ **have** $\text{NF}: \bigwedge u. u \triangleleft l \cdot \sigma \implies u \in \text{NF-terms } Q'$ **by** *auto*

from $lr \ R$ **have** $lr: (l, r) \in R'$ **by** *auto*

from $s \ t \ lr \ \text{NF} \ \text{nfs}$ **show** $(s, t) \in \text{rqrstep nfs}' \ Q' \ R'$ **by** *auto*

qed

lemma *rqrstep-mono*:

assumes $R: R \subseteq R'$ **and** $Q: \text{NF-terms } Q \subseteq \text{NF-terms } Q'$

shows $\text{rqrstep nfs } Q \ R \subseteq \text{rqrstep nfs } Q' \ R'$

by $(\text{rule } \text{rqrstep-all-mono}[\text{OF } R \ Q])$

lemma *nrqrstep-all-mono*:

assumes $\text{NF-terms } Q \subseteq \text{NF-terms } Q'$ **and** $R \subseteq R'$ **and** $Q \neq \{\}$ $\implies \text{nfs}' \implies$
 nfs

shows $\text{nrqrstep nfs } Q \ R \subseteq \text{nrqrstep nfs}' \ Q' \ R'$

proof

fix $s \ t$

assume $(s, t) \in \text{nrqrstep nfs } Q \ R$

then obtain $l \ r \ C \ \sigma$

where $\text{nf}: \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$

and $lr: (l, r) \in R$ **and** $\text{id}: C \neq \square \ s = C\langle l \cdot \sigma \rangle \ t = C\langle r \cdot \sigma \rangle$

and $\text{nfs}: \text{NF-subst nfs } (l, r) \ \sigma \ Q$

by *auto*

from *assms* nfs **have** $\text{nfs}: \text{NF-subst nfs}' (l, r) \ \sigma \ Q'$ **unfolding** *NF-subst-def* **by** *auto*

show $(s, t) \in \text{nrqrstep nfs}' \ Q' \ R'$

by $(\text{rule } \text{nrqrstepI}[\text{OF } - \text{ id}], \text{ insert nf nfs lr assms, unfold NF-subst-def, auto})$

qed

lemma *nrqrstep-mono*:

assumes $\text{NF-terms } Q \subseteq \text{NF-terms } Q'$ **and** $R \subseteq R'$

shows $nrqrstep\ nfs\ Q\ R \subseteq nrqrstep\ nfs\ Q'\ R'$
by (rule $nrqrstep\text{-all-mono}[OF\ assms]$)

lemma $nrqrstep\text{-imp-Fun-qstep}$: **assumes** $(s,t) \in nrqrstep\ nfs\ Q\ R$
shows $\exists f\ bef\ aft\ si\ ti. s = Fun\ f\ (bef\ @\ si\ \# \ aft) \wedge t = Fun\ f\ (bef\ @\ ti\ \# \ aft)$
 $\wedge (si,ti) \in qstep\ nfs\ Q\ R$

proof –
from $nrqrstepE[OF\ assms]$ **obtain** $l\ r\ C\ \sigma$ **where**
 $nf: \forall u \triangleleft l \cdot \sigma. u \in NF\text{-terms}\ Q$ **and** $lr: (l, r) \in R$ **and** $C: C \neq \square$
and $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$ **and** $nfs: NF\text{-subst}\ nfs\ (l, r)\ \sigma\ Q$.
from C **obtain** $f\ bef\ D\ aft$ **where** $C: C = More\ f\ bef\ D\ aft$ **by** (cases C , auto)
from $qstepI[OF\ nf\ lr\ refl\ refl\ nfs, of\ D]$ **show** ?thesis **unfolding** $s\ t\ C$ **by** auto
qed

lemma $qstep\text{-funas-term}$: **assumes** $wwf: wwf\text{-qtrs}\ Q\ R$
and $RF: funas\text{-trs}\ R \subseteq F$
and $sF: funas\text{-term}\ s \subseteq F$
and $step: (s,t) \in qstep\ nfs\ Q\ R$
shows $funas\text{-term}\ t \subseteq F$

proof –
from $qstepE[OF\ step]$ **obtain** $C\ \sigma\ l\ r$
where $nf: \forall u \triangleleft l \cdot \sigma. u \in NF\text{-terms}\ Q$ **and** $lr: (l, r) \in R$ **and** $s: s = C\langle l \cdot \sigma \rangle$
and $t: t = C\langle r \cdot \sigma \rangle$ **and** $nfs: NF\text{-subst}\ nfs\ (l, r)\ \sigma\ Q$.
from $only\text{-applicable-rules}[OF\ nf]\ lr\ wwf[unfolding\ wwf\text{-qtrs-def}]$ **have** $vars: vars\text{-term}\ r \subseteq vars\text{-term}\ l$ **by** auto
from $RF\ lr$ **have** $rF: funas\text{-term}\ r \subseteq F$
unfolding $funas\text{-trs-def}\ funas\text{-rule-def}\ [abs-def]$ **by** force
from sF **show** ?thesis **unfolding** $s\ t$ **using** $vars\ rF$ **by** (force simp: $funas\text{-term-subst}$)
qed

lemma $qsteps\text{-funas-term}$: **assumes** $wwf: wwf\text{-qtrs}\ Q\ R$
and $RF: funas\text{-trs}\ R \subseteq F$
and $sF: funas\text{-term}\ s \subseteq F$
and $steps: (s,t) \in (qstep\ nfs\ Q\ R)^*$
shows $funas\text{-term}\ t \subseteq F$
using $steps$

proof (induct)
case base
show ?case **by** fact
next
case (step $t\ u$)
from $qstep\text{-funas-term}[OF\ wwf\ RF\ step(3)\ step(2)]$ **show** ?case .
qed

lemma $nrqrstep\text{-funas-args-term}$: **assumes** $wwf: wwf\text{-qtrs}\ Q\ R$
and $RF: funas\text{-trs}\ R \subseteq F$
and $sF: funas\text{-args-term}\ s \subseteq F$
and $step: (s,t) \in nrqrstep\ nfs\ Q\ R$
shows $funas\text{-args-term}\ t \subseteq F$

proof –

from *nrqrstep-imp-Fun-qrstep*[*OF step*] **obtain**
f bef aft si ti **where** *s*: *s* = *Fun f (bef @ si # aft)* **and** *t*: *t* = *Fun f (bef @ ti # aft)* **and** *step*: (*si, ti*) ∈ *qrstep nfs Q R*
by *blast*
from *sF s* **have** *funas-term si* ⊆ *F* **unfolding** *funas-args-term-def* **by** *force*
from *qrstep-funas-term*[*OF wwf RF this step*] **show** *?thesis* **using** *sF* **unfolding**
s t funas-args-term-def **by** *force*
qed

lemma *qrstep-iff-rqrstep-or-nrqrstep*:

qrstep nfs Q R = *rqrstep nfs Q R* ∪ *nrqrstep nfs Q R* (**is** *?step* = *?root* ∪ *?nonroot*)

proof

show *?step* ⊆ *?root* ∪ *?nonroot*

proof

fix *s t* **assume** (*s, t*) ∈ *?step*

then obtain *C l r σ* **where** *NF*: ∀ *u* < *l* · *σ*. *u* ∈ *NF-terms Q* **and**

in-R: (*l, r*) ∈ *R* **and** *s*: *s* = *C*⟨*l* · *σ*⟩ **and** *t*: *t* = *C*⟨*r* · *σ*⟩

and *NF-subst nfs (l,r) σ Q* **by** *auto*

then show (*s, t*) ∈ *?root* ∪ *?nonroot* **by** (*cases C = □*) *auto*

qed

next

show *?root* ∪ *?nonroot* ⊆ *?step*

proof (*intro subrelI, elim UnE*)

fix *s t* **assume** (*s, t*) ∈ *rqrstep nfs Q R*

then obtain *l r σ* **where** *NF*: ∀ *u* < *l* · *σ*. *u* ∈ *NF-terms Q* **and**

in-R: (*l, r*) ∈ *R* **and** *s*: *s* = *□*⟨*l* · *σ*⟩ **and** *t*: *t* = *□*⟨*r* · *σ*⟩

and *NF-subst nfs (l,r) σ Q* **by** *auto*

then show (*s, t*) ∈ *qrstep nfs Q R* **using** *qrstepI[of l σ Q r R s □ t]* **by** *simp*

next

fix *s t* **assume** (*s, t*) ∈ *nrqrstep nfs Q R* **then show** (*s, t*) ∈ *qrstep nfs Q R*

by *auto*

qed

qed

lemma *qrstep-imp-ctxt-nrqrstep*:

assumes (*s,t*) ∈ *qrstep nfs Q R*

shows (*Fun f (bef @ s # aft)*, *Fun f (bef @ t # aft)*) ∈ *nrqrstep nfs Q R*

using *assms*

proof

fix *C σ l r*

assume *nf*: ∀ *u* < *l* · *σ*. *u* ∈ *NF-terms Q* **and**

lr: (*l, r*) ∈ *R*

and *s*: *s* = *C*⟨*l* · *σ*⟩ **and** *t*: *t* = *C*⟨*r* · *σ*⟩

and *nfs*: *NF-subst nfs (l,r) σ Q*

have *C*: *More f bef C aft* ≠ *□* **by** *auto*

show *?thesis*

```

unfolding nrqstep-def
apply (intro CollectI, unfold split)
apply (intro exI conjI)
  apply (rule nf)
  apply (rule lr)
  apply (rule C)
  by (auto simp: s t nfs)
qed

```

```

lemma grsteps-imp-ctxt-nrqsteps:
  assumes (s,t) ∈ (grstep nfs Q R)*
  shows (Fun f (bef @ s # aft), Fun f (bef @ t # aft)) ∈ (nrqstep nfs Q R)*
  using assms
proof (induct)
  case (step t u)
  from step(3) grstep-imp-ctxt-nrqstep[OF step(2), of f bef aft]
  show ?case by auto
qed simp

```

```

lemma ctxt-closed-nrqstep [intro]: ctxt.closed (nrqstep nfs Q R)
proof (rule one-imp-ctxt-closed)
  fix f bef s t aft
  assume (s,t) ∈ nrqstep nfs Q R
  from this[unfolded nrqstep-def] obtain l r C σ
    where NF: ∀ u ◁ l · σ. u ∈ NF-terms Q and lr: (l,r) ∈ R and C: C ≠ □
    and s: s = C⟨l · σ⟩ and t: t = C⟨r · σ⟩
    and nfs: NF-subst nfs (l,r) σ Q by auto
  show (Fun f (bef @ s # aft), Fun f (bef @ t # aft)) ∈ nrqstep nfs Q R
  proof (rule nrqstepI[OF NF lr - - nfs])
    show More f bef C aft ≠ □ by simp
  qed (insert s t, auto)
qed

```

```

lemma nrqstep-union: nrqstep nfs Q (R ∪ S) = nrqstep nfs Q R ∪ nrqstep nfs Q S
  unfolding nrqstep-def by blast

```

```

lemma nrqstep-imp-arg-grstep:
  assumes (s, t) ∈ nrqstep nfs Q R
  shows ∃ i < length (args s). (args s ! i, args t ! i) ∈ (grstep nfs Q R)
proof -
  from assms obtain l r C σ where NF: ∀ u ◁ l · σ. u ∈ NF-terms Q
    and in-R: (l, r) ∈ R and ne: C ≠ □ and s: s = C⟨l · σ⟩ and t: t = C⟨r · σ⟩
    and nfs: NF-subst nfs (l,r) σ Q by auto
  from ne obtain f ss1 D ss2 where C: C = More f ss1 D ss2 by (cases C) auto
  let ?i = length ss1
  show ?thesis
    by (rule exI[of - ?i], unfold s C t, insert in-R NF nfs, auto)
qed

```

lemma *nrqrstep-imp-arg-qrstps*:
assumes $(s, t) \in \text{nrqrstep nfs } Q \ R$
shows $(\text{args } s ! i, \text{args } t ! i) \in (\text{qrstep nfs } Q \ R)^\wedge =$
proof –
from *assms* **obtain** $l \ r \ C \ \sigma$ **where** $NF: \forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$
and *in-R*: $(l, r) \in R$ **and** *ne*: $C \neq \square$ **and** $s = C \langle l \cdot \sigma \rangle$ **and** $t = C \langle r \cdot \sigma \rangle$
and *nfs*: $\text{NF-subst nfs } (l, r) \ \sigma \ Q$ **by** *auto*
from *ne* **obtain** $f \ ss1 \ D \ ss2$ **where** $C: C = \text{More } f \ ss1 \ D \ ss2$ **by** (*cases C*) *auto*
let $?i = \text{length } ss1$
have $i < ?i \vee i = ?i \vee i > ?i$ **by** *auto*
then show *?thesis*
proof (*elim disjE*)
assume $i < ?i$
with *append-Cons-nth-left*[*OF this*] **show** *?thesis* **by** (*simp add: s t C*)
next
assume $i = ?i$
with *append-Cons-nth-middle*[*OF this*] **show** *?thesis* **using** $NF \text{ in-} R \ s \ t \ \text{nfs}$ **by**
(*auto simp: C*)
next
assume $i > ?i$
with *append-Cons-nth-right*[*OF this*] **show** *?thesis* **by** (*simp add: s t C*)
qed
qed

lemma *nrqrsteps-imp-arg-qrstps*:
assumes $(s, t) \in (\text{nrqrstep nfs } Q \ R)^*$ **shows** $(\text{args } s ! i, \text{args } t ! i) \in (\text{qrstep nfs } Q \ R)^*$
using *assms* **proof** (*induct rule: rtrancl-induct*)
case (*step u v*)
with *nrqrstep-imp-arg-qrstps*[*of u v nfs Q R i*] **show** *?case* **by** *auto*
qed *simp*

lemma *nrqrsteps-imp-arg-qrstps-count*:
assumes $(s, t) \in (\text{nrqrstep nfs } Q \ R)^\sim n$ **shows** $\exists m. m \leq n \wedge (\text{args } s ! i, \text{args } t ! i) \in (\text{qrstep nfs } Q \ R)^\sim m$
using *assms* **proof** (*induct n arbitrary: t*)
case (*Suc n u*)
from *Suc(2)* **obtain** t **where** $st: (s, t) \in (\text{nrqrstep nfs } Q \ R)^\sim n$ **and** $tu: (t, u) \in \text{nrqrstep nfs } Q \ R$ **by** *auto*
from *Suc(1)*[*OF st*] **obtain** m **where** $m: m \leq n$ **and** $st: (\text{args } s ! i, \text{args } t ! i) \in (\text{qrstep nfs } Q \ R)^\sim m$ **by** *auto*
from *nrqrstep-imp-arg-qrstps*[*OF tu, of i*] **have** $(\text{args } t ! i, \text{args } u ! i) \in \text{Id} \cup \text{qrstep nfs } Q \ R$ **(is ?tu ∈ -)** **by** *auto*
then show *?case*
proof
assume $?tu \in \text{Id}$ **with** $st \ m$ **show** *?case* **by** (*intro exI[of - m], auto*)
next
assume $?tu \in \text{qrstep nfs } Q \ R$ **with** $st \ m$ **show** *?case* **by** (*intro exI[of - Suc*

$m]$, *auto*)
qed
qed *simp*

lemma *nrqrstep-preserves-root*:
assumes $(s, t) \in \text{nrqrstep nfs } Q \ R$
shows $\text{root } s = \text{root } t$
using *assms*
proof (*standard*, *goal-cases*)
case ($1 \ l \ r \ C \ \sigma$)
hence $t: t = C\langle r \cdot \sigma \rangle$ **and** $C \neq \square$ **and** $s: s = C\langle l \cdot \sigma \rangle$ **by** *auto*
then obtain $f \ ss1 \ D \ ss2$ **where** $C = \text{More } f \ ss1 \ D \ ss2$ **by** (*cases* C) *auto*
then show *?thesis* **unfolding** $s \ t$ **by** *auto*
qed

lemma *nrqrsteps-preserve-root*:
assumes $(s, t) \in (\text{nrqrstep nfs } Q \ R)^*$
shows $\text{root } s = \text{root } t$
using *assms* **by** *induct* (*auto simp: nrqrstep-preserves-root*)

lemma *nrqrstep-preserves-root-fun*:
assumes *step*: $(\text{Fun } f \ ss, t) \in \text{nrqrstep nfs } Q \ R$
shows $\exists ts. t = \text{Fun } f \ ts$
using *nrqrstep-preserves-root*[*OF step*] **by** (*cases* t , *auto*)

lemma *nrqrsteps-preserve-root-fun*:
assumes *step*: $(\text{Fun } f \ ss, t) \in (\text{nrqrstep nfs } Q \ R)^*$
shows $\exists ts. t = \text{Fun } f \ ts$
using *nrqrsteps-preserve-root*[*OF step*] **by** (*cases* t , *auto*)

lemma *nrqrstep-num-args*:
assumes $(s, t) \in \text{nrqrstep nfs } Q \ R$ **shows** $\text{num-args } s = \text{num-args } t$
proof –
from *assms* **obtain** $l \ r \ C \ \sigma$ **where** $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$
and $(l, r) \in R$ **and** $C \neq \square$ **and** $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$ **by** *auto*
show *?thesis* **unfolding** $s \ t$ **by** (*cases* C) (*auto simp: $\langle C \neq \square \rangle$*)
qed

lemma *nrqrsteps-num-args*:
assumes $(s, t) \in (\text{nrqrstep nfs } Q \ R)^*$ **shows** $\text{num-args } s = \text{num-args } t$
using *assms* **by** (*induct rule: rtrancl.induct*) (*auto simp: nrqrstep-num-args*)

context
fixes *shp* :: $'f \Rightarrow 'f$
begin

interpretation *sharp-syntax* .

lemma *nrqrstep-imp-sharp-qrstep*:
assumes *step*: $(s, t) \in \text{nrqrstep nfs } Q \ R$
shows $(\# s, \# t) \in \text{nrqrstep nfs } Q \ R$
proof –
let $?qr = \text{qrstep nfs } Q \ R$
let $?nr = \text{nrqrstep nfs } Q \ R$
let $?Q = Q$
let $?R = R$
let $?Iqr = \text{nrqrstep nfs } ?Q \ ?R$
from *step* **obtain** $l \ r \ C \ \sigma$ **where** $C: C \neq \square$ **and** $lr: (l, r) \in R$ **and** $s: s = C\langle l \cdot \sigma \rangle$ **and** $t: t = C\langle r \cdot \sigma \rangle$
and $NF: \forall u \triangleleft l \cdot \sigma. \ u \in \text{NF-terms } Q \ \text{NF-subst nfs } (l, r) \ \sigma \ Q$
unfolding *nrqrstep-def rstep-r-c-s-def* **by** *auto*
from C **obtain** $D \ f \ \text{bef} \ \text{aft}$ **where** $C: C = \text{More } f \ \text{bef } D \ \text{aft}$ **by** (*cases C, auto*)
let $?Dl = D\langle l \cdot \sigma \rangle$
let $?Dr = D\langle r \cdot \sigma \rangle$
let $?C = \text{More } (\# f) \ \text{bef } \square \ \text{aft}$
from $s \ t \ C$ **have** $s: s = \text{Fun } f \ (\text{bef } @ \ ?Dl \ \# \ \text{aft})$ **and** $t: t = \text{Fun } f \ (\text{bef } @ \ ?Dr \ \# \ \text{aft})$ **by** *auto*
from $lr \ NF$ **have** $(?Dl, ?Dr) \in ?qr$ **by** *auto*
from *qrstep-imp-ctxt-nrqrstep* [*OF this*]
have $(?C\langle ?Dl \rangle, ?C\langle ?Dr \rangle) \in ?Iqr$ **by** *simp*
with $s \ t$ **show** *?thesis* **by** *auto*
qed

lemma *nrqrsteps-imp-sharp-qrsteps*:
assumes *step*: $(s, t) \in (\text{nrqrstep nfs } Q \ R)^*$
shows $(\# s, \# t) \in (\text{nrqrstep nfs } Q \ R)^*$
using *step*
proof (*induct rule: rtrancl-induct*)
case (*step u v*)
from *nrqrstep-imp-sharp-qrstep* [*OF step(2)*] **and** *step(3)*
show *?case* **by** *auto*
qed *simp*

end

lemma *qrstep-imp-nrqrstep*:
assumes *nvar*: $\forall (l, r) \in R. \ \text{is-Fun } l$
and *ndef*: $\neg \text{defined } (\text{applicable-rules } Q \ R) \ (\text{the } (\text{root } s))$
and *step*: $(s, t) \in \text{qrstep nfs } Q \ R$
shows $(s, t) \in \text{nrqrstep nfs } Q \ R$
proof –
from *step* [*unfolded qrstep-iff-rqrstep-or-nrqrstep*]
show *?thesis*
proof
assume $(s, t) \in \text{rqrstep nfs } Q \ R$
then **show** *?thesis*
proof

```

fix  $l\ r\ \sigma$ 
assume  $lr: (l,r) \in R$  and  $id: s = l \cdot \sigma$  and
   $nf: \forall\ u \triangleleft l \cdot \sigma. u \in NF\text{-terms}\ Q$ 
with  $nvar$  have  $is\text{-Fun}\ l$  by auto
then obtain  $f\ ll$  where  $l: l = Fun\ f\ ll$  by (cases  $l$ , auto)
with  $id$  obtain  $ss$  where  $s: s = Fun\ f\ ss$   $length\ ll = length\ ss$  by (cases  $s$ ,
auto)
from only-applicable-rules[ $OF\ nf$ ]  $lr$  have  $ulr: (l,r) \in applicable\text{-rules}\ Q\ R$ 
  unfolding applicable-rules-def by auto
from  $id[unfolding\ l\ s]$   $ndef[unfolding\ s]$   $ulr[unfolding\ l]$ 
have False unfolding defined-def by force
then show ?thesis by simp
qed
qed
qed

lemma qrsteps-imp-nrqrsteps:
assumes  $nvar: \forall\ (l, r) \in R. is\text{-Fun}\ l$ 
and  $ndef: \neg\ defined\ (applicable\text{-rules}\ Q\ R)\ (the\ (root\ s))$ 
and  $steps: (s,t) \in (qrstep\ nfs\ Q\ R)^*$ 
shows  $(s,t) \in (nrqrstep\ nfs\ Q\ R)^*$ 
using steps
proof (induct)
case (step  $t\ u$ )
have  $(t,u) \in nrqrstep\ nfs\ Q\ R$ 
proof (rule qrstep-imp-nrqrstep[ $OF\ nvar - step(2)$ ])
show  $\neg\ defined\ (applicable\text{-rules}\ Q\ R)\ (the\ (root\ t))$ 
using  $ndef$  unfolding nrqrsteps-num-args[ $OF\ step(3)$ ]
  nrqrsteps-preserve-root[ $OF\ step(3)$ ]
qed
with step(3) show ?case by auto
qed simp

lemma rel-qrsteps-imp-rel-nrqrsteps:
assumes  $nvar: \forall\ (l, r) \in R \cup Rw. is\text{-Fun}\ l$ 
and  $ndef: \neg\ defined\ (applicable\text{-rules}\ Q\ (R \cup Rw))\ (the\ (root\ s))$ 
and  $steps: (s,t) \in (qrstep\ nfs\ Q\ (R \cup Rw))^* \ O\ (qrstep\ nfs\ Q\ R)\ O\ (qrstep\ nfs\ Q\ (R \cup Rw))^*$ 
shows  $(s,t) \in (nrqrstep\ nfs\ Q\ (R \cup Rw))^* \ O\ nrqrstep\ nfs\ Q\ R\ O\ (nrqrstep\ nfs\ Q\ (R \cup Rw))^*$ 
proof –
let  $?Rw = qrstep\ nfs\ Q\ (R \cup Rw)$ 
let  $?R = qrstep\ nfs\ Q\ R$ 
let  $?Rwb = nrqrstep\ nfs\ Q\ (R \cup Rw)$ 
let  $?Rb = nrqrstep\ nfs\ Q\ R$ 
from steps obtain  $u\ v$  where  $su: (s,u) \in ?Rw^*$  and  $uv: (u,v) \in ?R$  and  $vt: (v,t) \in ?Rw^*$  by blast
from qrsteps-imp-nrqrsteps[ $OF\ nvar\ ndef\ su$ ] have  $su: (s,u) \in ?Rwb^*$  .

```

note $ndef = ndef[unfolds\ nrqrsteps\text{-}preserve\text{-}root[OF\ su]]$
then have $ndefR: \neg\ defined\ (applicable\text{-}rules\ Q\ R)\ (the\ (root\ u))$ **unfolding**
defined-def applicable-rules-def **by** *auto*
have $uv: (u,v) \in ?Rb$
by $(rule\ qrstep\text{-}imp\text{-}nrqrstep[OF\text{-} ndefR\ uv],\ insert\ nvar,\ auto)$
note $ndef = ndef[unfolds\ nrqrstep\text{-}preserves\text{-}root[OF\ uv]]$
from $qrsteps\text{-}imp\text{-}nrqrsteps[OF\ nvar\ ndef\ vt]$ **have** $vt: (v,t) \in ?Rwb^*$.
from $su\ uv\ vt$ **show** $?thesis$ **by** *auto*
qed

lemma $nrqrstep\text{-}nrrstep[simp]: nrqrstep\ nfs\ \{\} = nrrstep$
by $(intro\ ext,\ unfold\ nrrstep\text{-}def'\ nrqrstep\text{-}def,\ auto)$

lemma $args\text{-}steps\text{-}imp\text{-}nrqrsteps$:
assumes $steps: \bigwedge i. i < length\ ss \implies (ts\ !\ i,\ ss\ !\ i) \in (qrstep\ nfs\ Q\ R)^*$
and $len: length\ ts = length\ ss$
shows $(Fun\ f\ ts,\ Fun\ f\ ss) \in (nrqrstep\ nfs\ Q\ R)^*$
by $(rule\ args\text{-}steps\text{-}imp\text{-}steps\text{-}gen[OF\ qrsteps\text{-}imp\text{-}ctxt\text{-}nrqrsteps\ len\ steps],\ auto)$

lemma $qrsteps\text{-}rqrstep\text{-}cases\text{-}n$:
assumes $(Fun\ f\ ss,\ t) \in (qrstep\ nfs\ Q\ R) \sim^n$
shows $(\exists\ ts.$
 $\quad length\ ts = length\ ss \wedge$
 $\quad t = Fun\ f\ ts \wedge$
 $\quad (\forall i < length\ ss. \exists m \leq n. (ss\ !\ i,\ ts\ !\ i) \in (qrstep\ nfs\ Q\ R) \sim^m) \wedge$
 $\quad (Fun\ f\ ss,\ t) \in (nrqrstep\ nfs\ Q\ R) \sim^n)$
 $\quad \vee (\exists m1 < n. \exists m2 < n. (Fun\ f\ ss,\ t) \in (nrqrstep\ nfs\ Q\ R) \sim^{m1} O (rqrstep\ nfs$
 $\quad Q\ R) O (qrstep\ nfs\ Q\ R) \sim^{m2})$
using *assms* **proof** $(induct\ n\ arbitrary: t)$
case 0 **then show** $?case$ **by** *auto*
next
let $?QR = qrstep\ nfs\ Q\ R$
let $?RQR = rqrstep\ nfs\ Q\ R$
let $?NRQR = nrqrstep\ nfs\ Q\ R$
case $(Suc\ m)$
then obtain u **where** $su: (Fun\ f\ ss,\ u) \in ?QR \sim^m$ **and** $ut: (u,\ t) \in ?QR$ **by**
auto
from $Suc(1)[OF\ su]$ **show** $?case$
proof
assume $\exists m1 < m. \exists m2 < m. (Fun\ f\ ss,\ u) \in ?NRQR \sim^{m1} O ?RQR O$
 $?QR \sim^{m2}$
then obtain $v\ m1\ m2$ **where** $sv: (Fun\ f\ ss,\ v) \in ?NRQR \sim^{m1} O ?RQR \wedge m1$
 $< m$ **and** $(v,\ u) \in ?QR \sim^{m2} \wedge m2 < m$ **by** *auto*
with $\langle (u,\ t) \in ?QR \rangle$ **have** $(v,\ t) \in ?QR \sim^{(Suc\ m2)} \wedge Suc\ m2 < Suc\ m$ **by**
auto
with sv **have** $(Fun\ f\ ss,\ t) \in ?NRQR \sim^{m1} O ?RQR O ?QR \sim^{Suc\ m2} \wedge m1$
 $< Suc\ m \wedge Suc\ m2 < Suc\ m$ **by** *auto*
then show $?thesis$ **by** *metis*

```

next
  assume  $\exists us. \text{length } us = \text{length } ss \wedge u = \text{Fun } f \text{ } us \wedge (\forall i < \text{length } ss. \exists k \leq m. (ss!i, us!i) \in ?QR \sim^k) \wedge (\text{Fun } f \text{ } ss, u) \in ?NRQR \sim^m$ 
  then obtain  $us$  where  $\text{len}: \text{length } us = \text{length } ss$  and  $u: u = \text{Fun } f \text{ } us$ 
    and  $\text{isteps}: \forall i < \text{length } ss. \exists k \leq m. (ss!i, us!i) \in ?QR \sim^k$  and  $\text{nrsteps}: (\text{Fun } f \text{ } ss, u) \in ?NRQR \sim^m$  by auto
  from  $ut$  have  $(u, t) \in ?RQR \vee (u, t) \in ?NRQR$  unfolding  $\text{qrstep-iff-rqrstep-or-nrqrstep}$ 
by simp
  then show  $?thesis$ 
proof
  assume  $(u, t) \in ?RQR$ 
  with  $\langle (\text{Fun } f \text{ } ss, u) \in ?NRQR \sim^m \rangle$  have  $(\text{Fun } f \text{ } ss, t) \in ?NRQR \sim^m \text{ } O \text{ } ?RQR$ 
by blast
  then have  $(\text{Fun } f \text{ } ss, t) \in ?NRQR \sim^m \text{ } O \text{ } ?RQR \text{ } O \text{ } ?QR \sim^0$  by auto
  then show  $?thesis$  by blast
next
  assume  $ut: (u, t) \in ?NRQR$ 
  then have  $st: (\text{Fun } f \text{ } ss, t) \in ?NRQR \sim \text{Suc } m$  using  $\text{nrsteps}$  by auto
  have  $*$ :  $(\text{Fun } f \text{ } ss, t) \in ?NRQR^*$  by (rule  $\text{relpow-imp-rtranc1}[OF \text{ } st]$ )
  {
    fix  $i$ 
    assume  $i < \text{length } ss$ 
    with  $\text{isteps}$  obtain  $k$  where  $k: k \leq m$  and  $ssus: (ss!i, us!i) \in ?QR \sim^k$ 
by auto
    from  $\text{nrqrstep-imp-arg-qrsteps}[OF \text{ } ut]$   $u$  have  $(us!i, \text{args } t!i) \in ?QR \sim^=$ 
by auto
    then have  $us!i = \text{args } t!i \vee (us!i, \text{args } t!i) \in ?QR$  by auto
    then have  $\exists k \leq \text{Suc } m. (ss!i, \text{args } t!i) \in ?QR \sim^k$ 
    proof
      assume  $us!i = \text{args } t!i$ 
      then have  $(ss!i, \text{args } t!i) \in ?QR \sim^k$  using  $ssus$  by auto
      then show  $?thesis$  using  $k \text{ le-SucI}$  by fast
    next
      assume  $(us!i, \text{args } t!i) \in ?QR$ 
      with  $ssus$  have  $(ss!i, \text{args } t!i) \in ?QR \sim (\text{Suc } k)$  by auto
      then show  $?thesis$  using  $k \text{ Suc-le-mono}$  by blast
    qed
  }
  then show  $?thesis$  using  $\text{nrqrsteps-num-args}[OF *]$   $\text{nrqrsteps-preserve-root-fun}[OF *]$   $st$  by auto
qed
qed
qed
qed

lemma  $\text{qrsteps-rqrstep-cases-nrqrstep}$ :
  assumes  $(\text{Fun } f \text{ } ss, t) \in (\text{qrstep nfs } Q \text{ } R)^*$ 
  shows  $(\exists ts. \text{length } ts = \text{length } ss \wedge t = \text{Fun } f \text{ } ts \wedge$ 

```

```

    (∀ i < length ss. (ss ! i, ts ! i) ∈ (qrstep nfs Q R)*))
    ∨ (Fun f ss, t) ∈ (nrqrstep nfs Q R)* O (qrstep nfs Q R) O (qrstep nfs Q R)*
using assms proof (induct)
  case base then show ?case by auto
next
  case (step u t)
  then have (∃ ts. length ts = length ss ∧ u = Fun f ts
    ∧ (∀ i < length ss. (ss ! i, ts ! i) ∈ (qrstep nfs Q R)*))
    ∨ (Fun f ss, u) ∈ (nrqrstep nfs Q R)* O qrstep nfs Q R O (qrstep nfs Q R)*
by simp
  then show ?case
  proof
    assume (Fun f ss, u) ∈ (nrqrstep nfs Q R)* O qrstep nfs Q R O (qrstep nfs Q
R)*
    then obtain v where (Fun f ss, v) ∈ (nrqrstep nfs Q R)* O qrstep nfs Q R
    and (v, u) ∈ (qrstep nfs Q R)* by auto
    with ⟨(u, t) ∈ qrstep nfs Q R⟩ have (v, t) ∈ (qrstep nfs Q R)* by auto
    with ⟨(Fun f ss, v) ∈ (nrqrstep nfs Q R)* O qrstep nfs Q R⟩ show ?thesis by
auto
  next
    assume ∃ ts. length ts = length ss ∧ u = Fun f ts
      ∧ (∀ i < length ss. (ss ! i, ts ! i) ∈ (qrstep nfs Q R)*))
    then obtain ts where len: length ts = length ss and u: u = Fun f ts
      and steps: ∀ i < length ss. (ss ! i, ts ! i) ∈ (qrstep nfs Q R)* by auto
    from len steps have (Fun f ss, u) ∈ (nrqrstep nfs Q R)* unfolding u
      by (simp add: args-steps-imp-nrqrsteps)
    from ⟨(u, t) ∈ qrstep nfs Q R⟩ have (u, t) ∈ qrstep nfs Q R ∨ (u, t) ∈ nrqrstep
nfs Q R
    unfolding qrstep-iff-qrstep-or-nrqrstep by simp
    then show ?thesis
    proof
      assume (u, t) ∈ qrstep nfs Q R
      with ⟨(Fun f ss, u) ∈ (nrqrstep nfs Q R)*⟩ have (Fun f ss, t) ∈ (nrqrstep nfs
Q R)* O qrstep nfs Q R
      by auto
      then show ?thesis by auto
    next
      assume (u, t) ∈ nrqrstep nfs Q R
      from nrqrstep-preserves-root[OF this [unfolded u]]
      obtain us where t: t = Fun f us by (cases t, auto)
      from nrqrstep-num-args[OF ⟨(u, t) ∈ nrqrstep nfs Q R⟩]
      have num-args u = num-args t by simp
      then have length ts = length us unfolding u t by simp
      with len have len': length ss = length ts by simp
      from nrqrstep-imp-arg-qrsteps[OF ⟨(u, t) ∈ nrqrstep nfs Q R⟩]
      have ∀ i < length ts. (ts ! i, args t ! i) ∈ (qrstep nfs Q R)^ unfolding u by
auto
      then have ∀ i < length ts. (ts ! i, us ! i) ∈ (qrstep nfs Q R)^ unfolding t by
simp

```

then have *next-steps*: $\forall i < \text{length } ts. (ts!i, us!i) \in (\text{qrstep nfs } Q \ R)^*$ **by** *auto*
have $\text{length } us = \text{length } ss$ **using** $\langle \text{length } ts = \text{length } us \rangle$ **and** *len* **by** *simp*
moreover have $t = \text{Fun } f \ us$ **by** *fact*
moreover have $\forall i < \text{length } ss. (ss!i, us!i) \in (\text{qrstep nfs } Q \ R)^*$
proof (*intro allI impI*)
 fix *i* **assume** $i < \text{length } ss$
 with steps have $(ss!i, ts!i) \in (\text{qrstep nfs } Q \ R)^*$ **by** *simp*
 moreover from $\langle i < \text{length } ss \rangle$ **and next-steps have** $(ts!i, us!i) \in (\text{qrstep}$
nfs } Q \ R)^
 unfolding *len'* **by** *simp*
 ultimately show $(ss!i, us!i) \in (\text{qrstep nfs } Q \ R)^*$ **by** *auto*
 qed
 ultimately show *?thesis* **by** *auto*
qed
qed
qed*

lemma *qrsteps-rqrstep-cases*:
 assumes $(\text{Fun } f \ ss, t) \in (\text{qrstep nfs } Q \ R)^*$
 shows $\exists ts. \text{length } ts = \text{length } ss \wedge$
 $t = \text{Fun } f \ ts \wedge$
 $(\forall i < \text{length } ss. (ss!i, ts!i) \in (\text{qrstep nfs } Q \ R)^*)$
 $\vee (\text{Fun } f \ ss, t) \in (\text{qrstep nfs } Q \ R)^* \ O \ (\text{rqrstep nfs } Q \ R) \ O \ (\text{qrstep nfs } Q \ R)^*$
proof –
 have $(\text{nrqrstep nfs } Q \ R)^* \subseteq (\text{qrstep nfs } Q \ R)^*$ **unfolding** *qrstep-iff-rqrstep-or-nrqrstep*
 by *regexp*
 then show *?thesis*
 using *qrsteps-rqrstep-cases-nrqrstep[OF assms]* **by** *auto*
qed

lemma *nondef-root-imp-arg-qrsteps*:
 assumes *steps*: $(\text{Fun } f \ ss, t) \in (\text{qrstep nfs } Q \ R)^*$
 and vars: $\forall (l, r) \in R. \text{is-Fun } l$
 and ndef: $\neg \text{defined } R \ (f, \text{length } ss)$
 shows $\exists ts. \text{length } ts = \text{length } ss \wedge t = \text{Fun } f \ ts \wedge (\forall i < \text{length } ss. (ss!i, ts!i) \in (\text{qrstep nfs } Q \ R)^*)$
proof –
 from *qrsteps-rqrstep-cases[OF steps]*
 show *?thesis*
proof
 assume $(\text{Fun } f \ ss, t) \in (\text{qrstep nfs } Q \ R)^* \ O \ \text{rqrstep nfs } Q \ R \ O \ (\text{qrstep nfs } Q \ R)^*$
 then obtain *u v* **where** $su: (\text{Fun } f \ ss, u) \in (\text{qrstep nfs } Q \ R)^*$ **and** $uv: (u, v) \in \text{rqrstep nfs } Q \ R$ **by** *auto*
 from *first-step[OF - su uv, unfolded qrstep-iff-rqrstep-or-nrqrstep, OF Un-commute]*
 obtain *u v* **where** $su: (\text{Fun } f \ ss, u) \in (\text{nrqrstep nfs } Q \ R)^*$ **and** $uv: (u, v) \in \text{rqrstep nfs } Q \ R$ **by** *auto*

```

    from nrqrsteps-preserve-root[OF su] nrqrsteps-num-args[OF su] obtain us
  where u: u = Fun f us
    and us: length ss = length us by (cases u, auto)
    from uv[unfolded u] rqrstep-def obtain l r σ where lr: (l,r) ∈ R and fus: Fun
  f us = l · σ by auto
    from vars[THEN bspec[OF - lr]] fus obtain ls where l = Fun f ls by (cases
  l, auto)
    with fus lr have defined R (f,length ss) unfolding us defined-def by auto
    with ndef
    show ?thesis by auto
  qed simp
qed

```

lemmas args-qrstps-imp-qrstps = args-steps-imp-steps[OF ctxt-closed-qrstps]

lemmas qrstep-imp-map-rstep = rstep-imp-map-rstep[OF qrstep-into-rstep]

lemmas qrsteps-imp-map-rsteps = rsteps-imp-map-rsteps[OF qrsteps-into-rsteps]

lemma NF-imp-subt-NF: assumes $t \in \text{NF-terms } Q$ shows $\forall u \triangleleft t. u \in \text{NF-terms } Q$

proof(intro allI impI)

fix u

assume t: $t \triangleright u$

{

fix v

assume uv: $(u,v) \in \text{rstep } (\text{Id-on } Q)$

from t obtain C where $t = C\langle u \rangle$ by auto

from uv have $(t, C\langle v \rangle) \in \text{rstep } (\text{Id-on } Q)$ unfolding t by auto

with assms have False by auto

}

thus $u \in \text{NF-terms } Q$ by auto

qed

lemma qrstep-diff: assumes $R \subseteq S$ shows $\text{qrstep nfs } Q \ R - \text{qrstep nfs } Q \ (D \cap S) \subseteq \text{qrstep nfs } Q \ (R - D)$ (is ?L \subseteq ?R)

proof -

{

fix s t

assume $(s,t) \in ?L$

then have yes: $(s,t) \in \text{qrstep nfs } Q \ R$ and no: $(s,t) \notin \text{qrstep nfs } Q \ (D \cap S)$

by auto

from yes no have $(s,t) \in ?R$

unfolding qrstep-rule-conv[where $R = R$]

unfolding qrstep-rule-conv[where $R = D \cap S$]

unfolding qrstep-rule-conv[where $R = R - D$]

using assms

by blast

}

then show ?thesis by auto

qed

lemma *wwf-qtrs-qstep-Fun*:

assumes *wwf-qtrs* $Q\ R$ **and** $(s, t) \in \text{qstep nfs } Q\ R$

shows $\exists f\ ss. s = \text{Fun } f\ ss$

proof –

from *assms* **obtain** $C\ \sigma\ l\ r$ **where** $(l, r) \in R$ **and** $s = C\langle l \cdot \sigma \rangle$ **by** *auto*

from *wwf-qtrs-imp-left-fun* [*OF* *assms*(1) $\langle (l, r) \in R \rangle$]

obtain $f\ ls$ **where** $l = \text{Fun } f\ ls$ **by** *auto*

show *?thesis* **unfolding** $\langle s = C\langle l \cdot \sigma \rangle \rangle\ l$ **by** (*induct* C) *simp-all*

qed

lemma *qstep-preserves-undefined-root*:

assumes *ndef*: $\neg \text{defined (applicable-rules } Q\ R)$ (*the (root s)*)

and *nvar*: $\forall (l, r) \in R. \text{is-Fun } l$

and *step*: $(s, t) \in \text{qstep nfs } Q\ R$

shows $\neg \text{defined (applicable-rules } Q\ R)$ (*the (root t)*)

proof –

from *qstep-imp-nrqstep* [*OF* *nvar* *ndef* *step*] **have** *step*: $(s, t) \in \text{nrqstep nfs } Q\ R$

.

from *ndef* [*unfolded nrqstep-preserves-root* [*OF* *step*] *nrqstep-num-args* [*OF* *step*]]

show *?thesis* .

qed

lemma *qsteps-preserve-undefined-root*:

assumes *ndef*: $\neg \text{defined (applicable-rules } Q\ R)$ (*the (root s)*)

and *nvar*: $\forall (l, r) \in R. \text{is-Fun } l$

and *step*: $(s, t) \in (\text{qstep nfs } Q\ R)^*$

shows $\neg \text{defined (applicable-rules } Q\ R)$ (*the (root t)*)

proof –

from *qsteps-imp-nrqsteps* [*OF* *nvar* *ndef* *step*] **have** *step*: $(s, t) \in (\text{nrqstep nfs } Q\ R)^*$

.

from *ndef* [*unfolded nrqsteps-preserve-root* [*OF* *step*] *nrqsteps-num-args* [*OF* *step*]]

show *?thesis* .

qed

lemma *wf-trs-imp-wwf-qtrs*:

assumes *wf-trs* R **shows** *wwf-qtrs* $Q\ R$

using *assms* **by** (*auto simp: wwf-qtrs-def wf-trs-def*)

lemma *SN-on-imp-qstep-wf-rules*:

assumes *SN-on* (*qstep nfs* $Q\ R$) $\{s\}$ **and** $(s, t) \in \text{qstep nfs } Q\ R$ **and** *nfs*: $\neg \text{nfs}$

shows $(s, t) \in \text{qstep nfs } Q\ (\text{wf-rules } R)$

using $\langle (s, t) \in \text{qstep nfs } Q\ R \rangle$

proof

fix $C\ \sigma\ l\ r$

assume *NF-terms*: $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$

and $(l, r) \in R$

```

    and  $s: s = C\langle l \cdot \sigma \rangle$  and  $t: t = C\langle r \cdot \sigma \rangle$ 
  show ?thesis
  proof (cases  $(l, r) \in wf\text{-rules } R$ )
    case True then show ?thesis unfolding  $s\ t$  using  $NF\text{-terms } nfs$  by auto
  next
    case False
    with  $\langle (l, r) \in R \rangle$ 
      have  $is\text{-Var } l \vee (\exists x. x \in vars\text{-term } r - vars\text{-term } l)$ 
      unfolding  $wf\text{-rules-def } wf\text{-rule-def}$  by auto
    then show ?thesis
    proof
      assume  $is\text{-Var } l$ 
      then obtain  $x$  where  $l: l = Var\ x$  by auto
      from  $left\text{-Var-imp-not-SN-grstep}[OF\ \langle (l, r) \in R \rangle [unfolded\ l]]$ 
        and  $assms$  show ?thesis by simp
    next
      assume  $\exists x. x \in vars\text{-term } r - vars\text{-term } l$ 
      then obtain  $x$  where  $x \in vars\text{-term } r - vars\text{-term } l$ 
        and  $empty: vars\text{-term } l \cap \{x\} = \{\}$  by auto
      with  $SN\text{-on-imp-wwf-rule}[OF\ assms(1)\ s\ \langle (l, r) \in R \rangle\ NF\text{-terms } nfs]$ 
        and  $only\text{-applicable-rules}[OF\ NF\text{-terms}]$ 
      show ?thesis unfolding  $wwf\text{-rule-def}$  by auto
    qed
  qed
qed

```

lemma $SN\text{-on-imp-grsteps-wf-rules}$:

```

  assumes  $(s, t) \in (grstep\ nfs\ Q\ R)^*$  and  $SN\text{-on } (grstep\ nfs\ Q\ R)\ \{s\}$  and  $nfs: \neg$ 
   $nfs$ 
  shows  $(s, t) \in (grstep\ nfs\ Q\ (wf\text{-rules } R))^*$ 
  using  $\langle (s, t) \in (grstep\ nfs\ Q\ R)^* \rangle$ 
  proof (induct)
    case base show ?case ..
  next
    case (step  $u\ v$ )
    from  $steps\text{-preserve-SN-on}[OF\ \langle (s, u) \in (grstep\ nfs\ Q\ R)^* \rangle\ \langle SN\text{-on } (grstep\ nfs\ Q\ R)\ \{s\} \rangle]$ 
      have  $SN\text{-on } (grstep\ nfs\ Q\ R)\ \{u\}$  .
    from  $SN\text{-on-imp-grstep-wf-rules}[OF\ this\ \langle (u, v) \in grstep\ nfs\ Q\ R \rangle\ nfs]$ 
      have  $(u, v) \in grstep\ nfs\ Q\ (wf\text{-rules } R)$  .
    with  $\langle (s, u) \in (grstep\ nfs\ Q\ (wf\text{-rules } R))^* \rangle$  show ?case ..
  qed

```

lemmas $wwf\text{-qtrs-wf-rules} = wf\text{-trs-imp-wwf-qtrs}[OF\ wf\text{-trs-wf-rules}]$

lemmas $grstep\text{-wf-rules-subset} = grstep\text{-mono}[OF\ wf\text{-rules-subset}\ subset\text{-refl}]$

lemma $SN\text{-on-grstep-imp-SN-on-supt-union-grstep}$:

$SN\text{-on } (grstep\ nfs\ Q\ R)\ \{t\} \implies SN\text{-on } (\{\triangleright\} \cup grstep\ nfs\ Q\ R)\ \{t\}$

by (rule *SN-on-r-imp-SN-on-supt-union-r*[*OF ctxt-closed-qstep*])

lemma *supt-qstep-subset*:
 $\{\triangleright\} \circ \text{qstep nfs } Q \ R \subseteq \text{qstep nfs } Q \ R \circ \{\triangleright\}$
 using *supteq-qstep-subset*[of *nfs Q R*, unfolded *supteq-supt-set-conv*] by best

lemma *supt-Un-qstep-trancl-subset*:
 $(\{\triangleright\} \cup \text{qstep nfs } Q \ R)^+ \subseteq (\text{qstep nfs } Q \ R)^* \circ \{\triangleright\}$
 (is ?lhs \subseteq ?rhs)

proof
 fix *s t*
 assume $(s, t) \in ?lhs$
 then show $(s, t) \in ?rhs$
proof (induct *s t*)
 case (*r-into-trancl s t*)
 then show ?case by (rule *UnE*) auto
 next
 case (*trancl-into-trancl s t u*)
 from $\langle (t, u) \in \{\triangleright\} \cup \text{qstep nfs } Q \ R \rangle$
 show ?case
proof
 assume $t \triangleright u$
 with $\langle (s, t) \in (\text{qstep nfs } Q \ R)^* \circ \{\triangleright\} \rangle$
 show ?case using *supteq-trans*[of - *t u*] by auto
 next
 assume $(t, u) \in \text{qstep nfs } Q \ R$
 from $\langle (s, t) \in (\text{qstep nfs } Q \ R)^* \circ \{\triangleright\} \rangle$
 obtain *v* where $(s, v) \in (\text{qstep nfs } Q \ R)^*$ and $v \triangleright t$ by blast
 with $\langle (t, u) \in \text{qstep nfs } Q \ R \rangle$ have $(v, u) \in \{\triangleright\} \circ \text{qstep nfs } Q \ R$ by blast
 with *supteq-qstep-subset*[of *nfs Q R*]
 have $(v, u) \in \text{qstep nfs } Q \ R \circ \{\triangleright\}$ by blast
 with *rtrancl.rtrancl-into-rtrancl*[*OF* $\langle (s, v) \in (\text{qstep nfs } Q \ R)^* \rangle$]
 show ?case by blast
 qed
 qed
 qed

lemma *supteq-Un-qstep-trancl-subset*:
 $(\{\triangleright\} \cup \text{qstep nfs } Q \ R)^+ \subseteq (\text{qstep nfs } Q \ R)^* \circ \{\triangleright\}$
 (is ?lhs \subseteq ?rhs)

proof
 fix *s t*
 assume $(s, t) \in ?lhs$
 then show $(s, t) \in ?rhs$
proof (induct *s t*)
 case (*r-into-trancl s t*)
 then show ?case by (rule *UnE*) auto
 next
 case (*trancl-into-trancl s t u*)

```

from  $\langle (t, u) \in \{\triangleright\} \cup \text{qrstep nfs } Q \ R \rangle$ 
show ?case
proof
  assume  $t \sqsupseteq u$ 
  with  $\langle (s, t) \in (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\} \rangle$ 
    show ?case using supteq-trans[of - t u] by auto
next
  assume  $(t, u) \in \text{qrstep nfs } Q \ R$ 
  from  $\langle (s, t) \in (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\} \rangle$ 
    obtain  $v$  where  $(s, v) \in (\text{qrstep nfs } Q \ R)^*$  and  $v \sqsupseteq t$  by blast
  with  $\langle (t, u) \in \text{qrstep nfs } Q \ R \rangle$  have  $(v, u) \in \{\triangleright\} \ O \ \text{qrstep nfs } Q \ R$  by blast
  with supteq-qrstep-subset[of nfs Q R]
    have  $(v, u) \in \text{qrstep nfs } Q \ R \ O \ \{\triangleright\}$  by blast
  with rtrancl.rtrancl-into-rtrancl[OF  $\langle (s, v) \in (\text{qrstep nfs } Q \ R)^* \rangle$ ]
    show ?case by blast
qed
qed
qed

lemma supteq-Un-qrstep-rtrancl-subset:
   $(\{\triangleright\} \cup \text{qrstep nfs } Q \ R)^* \subseteq (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\}$ 
proof –
  from supteq-Un-qrstep-trancl-subset[of nfs Q R]
    have  $\text{Id} \cup (\{\triangleright\} \cup \text{qrstep nfs } Q \ R)^* \ O \ (\{\triangleright\} \cup \text{qrstep nfs } Q \ R) \subseteq (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\}$ 
  unfolding rtrancl-comp-trancl-conv
  unfolding supteq-supt-set-conv by auto
  then show ?thesis by auto
qed

lemma supt-Un-qrstep-subset:
   $\{\triangleright\} \cup \text{qrstep nfs } Q \ R \subseteq (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\}$ 
by auto

lemma supt-Un-qrstep-rtrancl-subset:
   $(\{\triangleright\} \cup \text{qrstep nfs } Q \ R)^* \subseteq (\text{qrstep nfs } Q \ R)^* \ O \ \{\triangleright\}$ 
  (is ?lhs  $\subseteq$  ?rhs)
proof
  fix  $s \ t$  assume  $(s, t) \in ?lhs$ 
  then show  $(s, t) \in ?rhs$ 
  proof (induct s t)
    case (rtrancl-refl t)
      show ?case by auto
  next
    case (rtrancl-into-rtrancl s t u)
      from  $\langle (t, u) \in \{\triangleright\} \cup \text{qrstep nfs } Q \ R \rangle$ 
        show ?case
  proof
    assume  $t \triangleright u$ 

```

```

    then have  $t \sqsupseteq u$  by auto
  with  $\langle (s, t) \in ?rhs \rangle$  show ?case
    using supseq-trans by blast
next
  assume  $(t, u) \in qstep\ nfs\ Q\ R$ 
  from  $\langle (s, t) \in ?rhs \rangle$ 
    obtain  $v$  where  $(s, v) \in (qstep\ nfs\ Q\ R)^*$  and  $v \sqsupseteq t$  by auto
  with  $\langle (t, u) \in qstep\ nfs\ Q\ R \rangle$  have  $(v, u) \in \{\sqsupseteq\} \circ qstep\ nfs\ Q\ R$  by auto
  with supseq-qstep-subset have  $(v, u) \in qstep\ nfs\ Q\ R \circ \{\sqsupseteq\}$  by blast
  with rtrancl.rtrancl-into-rtrancl[OF  $\langle (s, v) \in (qstep\ nfs\ Q\ R)^* \rangle$ ]
    show ?case by auto
qed
qed
qed

lemma qsteps-comp-supseq-subset:
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} \subseteq (\{\sqsupseteq\} \cup qstep\ nfs\ Q\ R)^*$ 
  by regexp

lemma qsteps-comp-supseq-subset':
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} \subseteq (\{\sqsupseteq\} \cup qstep\ nfs\ Q\ R)^+$ 
  by regexp

lemma qsteps-comp-supseq-supt-subset:
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} \subseteq (\{\sqsupset\} \cup qstep\ nfs\ Q\ R)^*$ 
  unfolding supseq-supt-set-conv by (regexp)

lemma qsteps-comp-supseq-conv:
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} = (\{\sqsupseteq\} \cup qstep\ nfs\ Q\ R)^+$ 
  using qsteps-comp-supseq-subset' supseq-Un-qstep-trancl-subset by blast

lemma qsteps-comp-supseq-conv':
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} = (\{\sqsupseteq\} \cup qstep\ nfs\ Q\ R)^*$ 
  using qsteps-comp-supseq-subset supseq-Un-qstep-rtrancl-subset by blast

lemma qsteps-comp-supseq-conv'':
   $(qstep\ nfs\ Q\ R)^* \circ \{\sqsupseteq\} = (\{\sqsupset\} \cup qstep\ nfs\ Q\ R)^*$ 
  using qsteps-comp-supseq-supt-subset supt-Un-qstep-rtrancl-subset by blast

lemmas supseq-Un-qstep-trancl-conv = qsteps-comp-supseq-conv[symmetric]
lemmas supseq-Un-qstep-rtrancl-conv = qsteps-comp-supseq-conv'[symmetric]
lemmas supt-Un-qstep-rtrancl-conv = qsteps-comp-supseq-conv''[symmetric]

lemma rhs-free-vars-imp-sig-qstep-not-SN-on:
  assumes  $R: (l, r) \in applicable\ rules\ Q\ R$  and free:  $\neg vars\ term\ r \subseteq vars\ term\ l$ 
  and  $F: funas\ trs\ R \subseteq F$ 
  and  $nfs: \neg nfs$ 
  shows  $\neg SN\ on\ (sig\ step\ F\ (qstep\ nfs\ Q\ R))\ \{l\}$ 

```

proof –

from *free* **obtain** x **where** $x: x \in \text{vars-term } r - \text{vars-term } l$ **by** *auto*
then have $x \in \text{vars-term } r$ **by** *simp*
from *supteq-Var*[*OF this*] **have** $r \supseteq \text{Var } x$.
then obtain C **where** $r: C\langle \text{Var } x \rangle = r$ **by** *auto*
let $?σ = λy. \text{ if } y = x \text{ then } l \text{ else } \text{Var } y$
let $?t = λi. ((C \cdot_c ?σ) \hat{i})\langle l \rangle$
from R **have** $R': (l, r) \in R$ **unfolding** *applicable-rules-def* **by** *auto*
from *rhs-wf*[*OF R' F*] **have** $\text{wf-r}: \text{funas-term } r \subseteq F$ **by** *fast*
from *lhs-wf*[*OF R' F*] **have** $\text{wf-l}: \text{funas-term } l \subseteq F$ **by** *fast*
from *wf-r*[*unfolded r[symmetric]*]
have $\text{wf-C}: \text{funas-ctxt } C \subseteq F$ **by** *simp*
from x **have** $\text{neg}: \forall y \in \text{vars-term } l. y \neq x$ **by** *auto*
have $l \cdot ?σ = l \cdot \text{Var}$
by (*rule term-subst-eq, insert neg, auto*)
then have $l \cdot ?σ = l$ **by** *simp*
have $\text{rsigma}: r \cdot ?σ = (C \cdot_c ?σ)\langle l \rangle$ **unfolding** *r[symmetric]* **by** *simp*
from wf-C **have** $\text{wf-C}: \text{funas-ctxt } (C \cdot_c ?σ) \subseteq F$ **using** wf-l **by** *auto*
show *?thesis*
proof (*rule wf-loop-imp-sig-ctxt-rel-not-SN*[*OF - wf-l wf-C ctxt-closed-qstep*])
show $(l, (C \cdot_c ?σ)\langle l \rangle) \in \text{qstep nfs } Q$ R
proof (*rule qstepI*[*OF - R', of ?σ - - Hole*], *unfold l rsigma*)
show $\forall u \triangleleft l. u \in \text{NF-terms } Q$ **using** R **unfolding** *applicable-rules-def applicable-rule-def* **by** *auto*
qed (*insert nfs, auto*)
qed
qed

lemma *lhs-var-imp-sig-qstep-not-SN*:

assumes *rule*: $(\text{Var } x, r) \in R$ **and** $F: \text{funas-trs } R \subseteq F$ **and** $\text{nfs}: \neg \text{nfs}$
shows $\neg \text{SN } (\text{sig-step } F (\text{qstep nfs } Q R))$
proof –
from *get-var-or-const*[*of r*]
obtain C t **where** $r: r = C\langle t \rangle$ **and** $\text{args}: \text{args } t = []$ **by** *auto*
from *rhs-wf*[*OF rule subset-refl*] F **have** $\text{wfr}: \text{funas-term } r \subseteq F$ **by** *auto*
from *wfr*[*unfolded r*] F
have $\text{wfC}: \text{funas-ctxt } C \subseteq F$ **and** $\text{wft}: \text{funas-term } t \subseteq F$ **by** *auto*
let $?σ = (\lambda x. t)$
from wfC wft **have** $\text{wfC}: \text{funas-ctxt } (C \cdot_c ?σ) \subseteq F$ **by** *auto*
have $\text{tsig}: t \cdot ?σ = t$ **using** args **by** (*cases t, auto*)
have $\neg \text{SN-on } (\text{sig-step } F (\text{qstep nfs } Q R)) \{t\}$
proof (*rule wf-loop-imp-sig-ctxt-rel-not-SN*[*OF - wft wfC ctxt-closed-qstep*])
show $(t, (C \cdot_c ?σ)\langle t \rangle) \in \text{qstep nfs } Q$ R
by (*rule qstepI*[*OF - rule, of ?σ - - Hole*], *unfold NF-terms-args-conv[symmetric]*)
 $r, \text{ auto simp: nfs tsig args}$
qed
then show *?thesis* **unfolding** *SN-def* **by** *auto*
qed

```

lemma SN-sig-grstep-imp-wwf-trs: assumes SN: SN (sig-step F (grstep nfs Q R))
and F: funas-trs R ⊆ F and nfs:  $\neg$  nfs
  shows wwf-qtrs Q R
proof (rule ccontr)
  assume  $\neg$  wwf-qtrs Q R
  then obtain l r where R:  $(l,r) \in \text{applicable-rules } Q R$ 
    and not-wf:  $(\forall f \text{ ts. } l \neq \text{Fun } f \text{ ts}) \vee \neg(\text{vars-term } r \subseteq \text{vars-term } l)$  unfolding
wwf-qtrs-def applicable-rules-def
    by auto
  from not-wf have  $\neg$  SN (sig-step F (grstep nfs Q R))
  proof
    assume free:  $\neg \text{vars-term } r \subseteq \text{vars-term } l$ 
    from rhs-free-vars-imp-sig-grstep-not-SN-on[OF R free F nfs] show ?thesis
unfolding SN-on-def by auto
  next
    assume  $\forall f \text{ ts. } l \neq \text{Fun } f \text{ ts}$ 
    then obtain x where l: l = Var x by (cases l) auto
    with R have  $(\text{Var } x, r) \in R$  unfolding l applicable-rules-def by simp
    from lhs-var-imp-sig-grstep-not-SN[OF this F nfs] show ?thesis .
  qed
with assms show False by blast
qed

```

```

lemma linear-term-weak-match-match:
  assumes linear-term t and weak-match s t
  shows  $\exists \sigma. s = t \cdot \sigma$ 
using assms
proof (induct t arbitrary: s)
  case (Var x) then show ?case by auto
next
  case (Fun f ts)
    note Fun' = this
    show ?case
    proof (cases s)
      case (Var y) show ?thesis using Fun by (simp add: Var)
    next
      case (Fun g ss)
        from Fun' have f = g and len: length ss = length ts by (simp add: Fun) +
        from Fun' have  $\forall i < \text{length } ts. \text{weak-match } (ss ! i) (ts ! i)$  by (simp add: len
Fun)
        with Fun' have  $\forall i. \exists \sigma. i < \text{length } ts \longrightarrow (ss ! i) = (ts ! i) \cdot \sigma$  by auto
        from choice[OF this] obtain  $\tau$  where substs:  $\forall i < \text{length } ts. ss ! i = (ts ! i) \cdot \tau i$ 
        by blast
        from subst-merge[OF Fun'(2)][unfolded linear-term.simps, THEN conjunct1]
        obtain  $\sigma$  where vars:  $\forall i < \text{length } ts. \forall x \in \text{vars-term } (ts ! i). \sigma x = \tau i x$  ..
        have  $\forall i < \text{length } ts. (ts ! i) \cdot \sigma = (ts ! i) \cdot \tau i$ 
        proof (intro allI impI)
          fix i assume i < length ts
          with vars have  $\forall x \in \text{vars-term } (ts ! i). \sigma x = \tau i x$  by simp

```

```

    then show  $(ts ! i) \cdot \sigma = (ts ! i) \cdot \tau i$  unfolding term-subst-eq-conv .
  qed
  with substs have  $\forall i < \text{length } ss. (ts ! i) \cdot \sigma = ss ! i$  by (simp add: len)
  with map-nth-eq-conv[OF len[symmetric], of  $\lambda t. t \cdot \sigma$ ]
    have map  $(\lambda t. t \cdot \sigma) ts = ss$  by simp
  then have  $s = \text{Fun } f ts \cdot \sigma$  by (simp add: Fun <f = g>)
  then show ?thesis by auto
  qed
  qed

```

```

lemma NF-terms-instance:
  assumes  $\forall s \triangleleft l. \sigma. s \in \text{NF-terms } Q$ 
  shows  $\forall s \triangleleft l. s \in \text{NF-terms } Q$ 
proof (intro allI impI)
  fix s assume  $l \triangleright s$ 
  then have  $l \cdot \sigma \triangleright s \cdot \sigma$  by (rule supt-subst)
  with assms have  $s \cdot \sigma \in \text{NF-terms } Q$  by simp
  from NF-instance[OF this] show  $s \in \text{NF-terms } Q$  .
qed

```

```

lemma NF-terms-ctxt:
  assumes  $\forall s \triangleleft C \langle l \rangle. s \in \text{NF-terms } Q$ 
  shows  $\forall s \triangleleft l. s \in \text{NF-terms } Q$ 
proof (intro impI allI)
  fix s assume  $l \triangleright s$ 
  from ctxt-imp-supteq[of  $C \ l$ ] have  $C \langle l \rangle \triangleright l$  .
  from this and  $\langle l \triangleright s \rangle$  have  $s \triangleleft C \langle l \rangle$  by (rule supteq-supt-trans)
  with assms show  $s \in \text{NF-terms } Q$  by simp
qed

```

```

lemma Q-subset-R-imp-same-NF:
  assumes subset:  $\text{NF-trs } R \subseteq \text{NF-terms } Q$ 
  and no-lhs-var:  $\bigwedge l r. nfs \implies (l, r) \in R \implies \text{is-Fun } l$ 
  shows  $\text{NF-trs } R = \text{NF } (qrstep \ nfs \ Q \ R)$ 
proof
  show  $\text{NF-trs } R \subseteq \text{NF } (qrstep \ nfs \ Q \ R)$ 
    by (rule NF-anti-mono, auto)
  next
  show  $\text{NF } (qrstep \ nfs \ Q \ R) \subseteq \text{NF-trs } R$ 
    proof
      fix t
      assume NFt:  $t \in \text{NF } (qrstep \ nfs \ Q \ R)$ 
      show  $t \in \text{NF-trs } R$ 
        proof (rule ccontr)
          assume  $t \notin \text{NF-trs } R$ 
          then obtain s where  $(t, s) \in rstep \ R$  by auto
          from rstep-imp-irstep[OF this no-lhs-var, of nfs] obtain s where step:  $(t, s) \in qrstep \ nfs \ (lhss \ R) \ R$  unfolding irstep-def by force

```

```

    have  $(t, s) \in \text{qrstep nfs } Q \ R$ 
    by (rule set-mp[OF qrstep-mono step], insert subset, auto)
    with NFt show False by auto
  qed
qed
qed

lemma all-ctxt-closed-qrstps[intro]: all-ctxt-closed  $F ((\text{qrstep nfs } Q \ R)^*)$ 
  by (rule trans-ctxt-imp-all-ctxt-closed[OF trans-rtrancl refl-rtrancl], blast)

lemma subst-qrstps-imp-qrstps:
  assumes  $\bigwedge x. x \in \text{vars-term } t \implies (\sigma \ x, \tau \ x) \in (\text{qrstep nfs } Q \ R)^*$ 
  shows  $(t \cdot \sigma, t \cdot \tau) \in (\text{qrstep nfs } Q \ R)^*$ 
  using assms
proof (induct t)
  case (Var x)
  then show ?case by auto
next
  case (Fun f ts)
  show ?case unfolding eval-term.simps
  proof (rule all-ctxt-closedD[of UNIV], unfold length-map)
    fix i
    assume i:  $i < \text{length } ts$ 
    from i have  $\sigma: \text{map } (\lambda t. t \cdot \sigma) \ ts \ ! \ i = ts \ ! \ i \cdot \sigma$  by auto
    from i have  $\tau: \text{map } (\lambda t. t \cdot \tau) \ ts \ ! \ i = ts \ ! \ i \cdot \tau$  by auto
    from i have mem:  $ts \ ! \ i \in \text{set } ts$  by auto
    have  $(ts \ ! \ i \cdot \sigma, ts \ ! \ i \cdot \tau) \in (\text{qrstep nfs } Q \ R)^*$ 
    by (rule Fun(1)[OF mem], insert mem Fun(2), auto)
    with  $\sigma \ \tau$  show  $(\text{map } (\lambda t. t \cdot \sigma) \ ts \ ! \ i, \text{map } (\lambda t. t \cdot \tau) \ ts \ ! \ i) \in (\text{qrstep nfs } Q \ R)^*$  by simp
  qed auto
qed

lemma subst-qrstps-imp-qrstps-at-pos:
  assumes  $p \in \text{poss } l$ 
  and  $\bigwedge x. x \in \text{vars-term } l \implies (\sigma \ x, \tau \ x) \in (\text{qrstep nfs } Q \ R)^*$ 
  shows  $(l \mid - \ p \cdot \sigma, l \mid - \ p \cdot \tau) \in (\text{qrstep nfs } Q \ R)^*$ 
proof -
  {
    fix y
    assume a:  $y \in \text{vars-term } (l \mid - \ p)$ 
    from supseq-trans[OF subseq-at-imp-supseq[OF assms(1)] vars-term-supseq(1)[OF a]]
    have  $y \in \text{vars-term } l$  using subseq-Var-imp-in-vars-term by fast
    with assms have  $(\sigma \ y, \tau \ y) \in (\text{qrstep nfs } Q \ R)^*$  by blast
  }
  then show ?thesis using subst-qrstps-imp-qrstps by blast
qed

```

lemma *instance-weak-match*:
 $s = t \cdot \sigma \implies \text{weak-match } s \ t$
by (induct $s \ t$ rule: *weak-match.induct*) *auto*

For linear terms, matching and weak matching are the same.

lemma *linear-term-weak-match-instance-conv*:
assumes *linear-term t*
shows $\text{weak-match } s \ t \longleftrightarrow (\exists \sigma. s = t \cdot \sigma)$
using *linear-term-weak-match-match*[*OF assms*] **and** *instance-weak-match* **by**
blast

lemma *qrsteps-rules-conv*:
 $((s, t) \in (\text{qrstep nfs } Q \ R)^*) = (\exists \ n \ ts \ lr. \ ts \ 0 = s \wedge ts \ n = t \wedge (\forall i < n. (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr \ i\} \wedge lr \ i \in R)) \ (\text{is } ?L = ?R)$
proof
assume $?L$
from *this*[*unfolded rtrancl-fun-conv*] **obtain** $n \ ts$ **where** *first*: $ts \ 0 = s$ **and** *last*:
 $ts \ n = t$ **and** *steps*: $\bigwedge i. i < n \implies (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ R$ **by** *auto*
 $\{$
 $\text{fix } i$
assume $i: i < n$
from *steps*[*OF this, unfolded qrstep-rule-conv*][**where** $R = R$]
have $\exists \ lr. lr \in R \wedge (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr\}$ **by** *auto*
 $\}$
then have $\forall i. \exists \ lr. i < n \longrightarrow lr \in R \wedge (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr\}$
by *auto*
from *choice*[*OF this*] **obtain** lr **where** $lr: \bigwedge i. i < n \implies lr \ i \in R$ **and** *steps*:
 $\bigwedge i. i < n \implies (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr \ i\}$ **by** *auto*
show $?R$ **using** *first last lr steps* **by** *blast*
next
assume $?R$
then obtain $n \ ts \ lr$ **where** *first*: $ts \ 0 = s$ **and** *last*: $ts \ n = t$
and *steps*: $\forall i < n. (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr \ i\} \wedge lr \ i \in R$
by *auto*
from *steps* **have** *steps*: $\forall i < n. (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ R$
unfolding *qrstep-rule-conv* [**where** $R = R$] **by** *auto*
show $?L$ **unfolding** *rtrancl-fun-conv* **using** *first last steps* **by** *blast*
qed

lemma *qrsteps-rules-conv'*:
assumes *left*: $((s, t) \in (\text{qrstep nfs } Q \ R)^* \ O \ \text{qrstep nfs } Q \ R' \ O \ (\text{qrstep nfs } Q \ R)^*)$
shows $(\exists \ n \ ts \ lr \ i. \ ts \ 0 = s \wedge ts \ n = t \wedge (\forall i < n. (ts \ i, ts \ (Suc \ i)) \in \text{qrstep nfs } Q \ \{lr \ i\} \wedge lr \ i \in R \cup R') \wedge i < n \wedge lr \ i \in R') \ (\text{is } ?Right)$
proof –
let $?Q = \text{qrstep nfs } Q$
let $?R = ?Q \ R$
let $?R' = ?Q \ R'$
show *thesis*

```

proof -
  from left
  obtain u v where su: (s, u) ∈ ?R* and uv: (u, v) ∈ ?R' and vt: (v, t) ∈ ?R*
by auto
  from su[unfolding grsteps-rules-conv]
  obtain lr ts n where first: ts 0 = s and last: ts n = u and steps: ∧ i. i < n
    ⇒ (ts i, ts (Suc i)) ∈ ?Q {lr i} and lr: ∧ i. i < n ⇒ lr i ∈ R
  by blast
  from vt[unfolding grsteps-rules-conv]
  obtain lr' ts' n' where first': ts' 0 = v and last': ts' n' = t and steps': ∧ i.
    i < n' ⇒ (ts' i, ts' (Suc i)) ∈ ?Q {lr' i} and lr': ∧ i. i < n' ⇒ lr' i ∈ R
  by blast
  from uv[unfolding grstep-rule-conv[where R = R']] obtain lr'' where lr'': lr''
    ∈ R' and uv: (u, v) ∈ grstep nfs Q {lr''} by auto
  let ?lr = λ i. if i < n then lr i else if i = n then lr'' else lr' (i - Suc n)
  let ?ts = λ i. if i ≤ n then ts i else ts' (i - Suc n)
  let ?n = Suc (n + n')
  show ?Right
proof (intro exI conjI)
  show ∀ i < ?n. (?ts i, ?ts (Suc i)) ∈ ?Q {?lr i} ∧ ?lr i ∈ R ∪ R'
proof (intro allI impI)
  fix i
  assume i: i < ?n
  show (?ts i, ?ts (Suc i)) ∈ ?Q {?lr i} ∧ ?lr i ∈ R ∪ R'
proof (cases i < n)
  case True
  with steps lr show ?thesis by simp
next
  case False
  show ?thesis
proof (cases i = n)
  case True
  with uv lr'' first' last show ?thesis by simp
next
  case False
  with ⟨¬ i < n⟩ have i > n by auto
  then have i = Suc n + (i - Suc n) by auto
  then obtain k where i': i = Suc n + k by auto
  with i have k: k < n' by auto
  from steps'[OF k] lr'[OF k] show ?thesis unfolding i'
  by simp
qed
qed
qed
next
  show ?ts 0 = s using first by auto
next
  show ?ts ?n = t using last' by auto
next

```

```

    show  $n < ?n$  by auto
  next
    show  $?lr\ n \in R'$  using  $lr''$  by auto
  qed
qed
qed

lemma qrsteps-rules-conv'':
  assumes first:  $ts\ 0 = s$ 
    and last:  $ts\ n = t$ 
    and steps:  $\bigwedge i. i < n \implies (ts\ i, ts\ (Suc\ i)) \in qrstep\ nfs\ Q\ \{lr\ i\}$ 
    and lr:  $\bigwedge i. i < n \implies lr\ i \in R' \cup R$ 
    and i:  $i < n$ 
    and i':  $lr\ i \in R'$ 
  shows  $(s, t) \in (qrstep\ nfs\ Q\ (R' \cup R))^* \ O\ qrstep\ nfs\ Q\ R' \ O\ (qrstep\ nfs\ Q\ (R' \cup R))^*$ 
proof -
  let  $?Q = qrstep\ nfs\ Q$ 
  let  $?R = ?Q\ (R' \cup R)$ 
  let  $?R' = ?Q\ R'$ 
  have one:  $(s, ts\ i) \in ?R^*$ 
    unfolding qrsteps-rules-conv using first steps lr i
    by (intro exI[of - i] exI[of - ts] exI[of - lr], auto)
  have two:  $(ts\ i, ts\ (Suc\ i)) \in ?R'$ 
    unfolding qrstep-rule-conv[where  $R = R'$ ] using steps[OF i] i' by auto
  from i have n:  $n = (n - Suc\ i) + Suc\ i$  by auto
  then obtain k where n:  $n = k + Suc\ i$  by auto
  have three:  $(ts\ (Suc\ i), t) \in ?R^*$ 
    unfolding qrsteps-rules-conv using insert last lr steps n
    by (intro exI[of - k] exI[of - shift ts (Suc i)] exI[of - shift lr (Suc i)], auto)
  from one two three
  show thesis by auto
qed

```

```

lemma normalize-subst-qrsteps-inn-partial:
  fixes  $Q\ R\ R'\ \sigma\ nfs\ b$ 
  defines  $\tau: \tau \equiv \lambda x. \text{if } b\ x \text{ then some-NF } (qrstep\ nfs\ Q\ R')\ (\sigma\ x) \text{ else } \sigma\ x$ 
  assumes UNF:  $UNF\ (qrstep\ nfs\ Q\ R')$ 
  and R':  $\bigwedge x\ u. x \in \text{vars-term } t \implies b\ x \implies (\sigma\ x, u) \in (qrstep\ nfs\ Q\ R)^* \implies (\sigma\ x, u) \in (qrstep\ nfs\ Q\ R')^*$ 
  and steps:  $(t \cdot \sigma, s) \in (qrstep\ nfs\ Q\ R)^*$ 
  and s:  $s \in \text{NF-terms } Q$ 
  and inn:  $\text{NF-terms } Q \subseteq \text{NF-trs } R'$ 
  shows  $(\forall x \in \text{vars-term } t. (\sigma\ x, \tau\ x) \in (qrstep\ nfs\ Q\ R')^* \wedge (b\ x \longrightarrow \tau\ x \in \text{NF-terms } Q)) \wedge (t \cdot \tau, s) \in (qrstep\ nfs\ Q\ R)^*$ 
  using steps s R'
proof (induct t arbitrary: s)
  case (Var x)
  let  $?QR = qrstep\ nfs\ Q\ R$ 

```

```

let ?QR' = qrstep nfs Q R'
from Var(2) inn have sNF:  $s \in NF\text{-trs } R'$  by auto
with NF-anti-mono[OF qrstep-mono[OF subset-refl, of Q {} nfs R']]
have sNF':  $s \in NF \text{ ?QR'}$  by auto
show ?case
proof (cases b x)
  case True
  from Var(1) Var(3)[OF - True] have steps:  $(\sigma x, s) \in \text{?QR}'^*$  by auto
  from some-NF-UNF[OF UNF steps sNF'] have s:  $s = \tau x$  unfolding  $\tau$  using
True by auto
  then show ?thesis using steps Var(2) by auto
next
  case False
  then show ?thesis using steps Var unfolding  $\tau$  by auto
qed
next
case (Fun f ts)
let ?QR = qrstep nfs Q R
let ?QR' = qrstep nfs Q R'
let ?P =  $\lambda ss i. (\forall x \in \text{vars-term } (ts ! i). (\sigma x, \tau x) \in \text{?QR}'^* \wedge (b x \longrightarrow \tau x \in$ 
NF-terms Q))  $\wedge (ts ! i \cdot \tau, ss ! i) \in \text{?QR}^*$ 
{
  fix ss
  assume len:  $\text{length } ss = \text{length } ts$ 
  and steps:  $\bigwedge i. i < \text{length } ts \implies (ts ! i \cdot \sigma, ss ! i) \in \text{?QR}^*$ 
  and NF:  $\bigwedge i. i < \text{length } ts \implies ss ! i \in \text{NF-terms } Q$ 
  let ?p = ?P ss
  {
    fix i
    assume i:  $i < \text{length } ts$ 
    with len have mem:  $ts ! i \in \text{set } ts$  by auto
    have ?p i
      by (rule Fun(1)[OF mem steps[OF i] NF[OF i] Fun(4)], insert mem, auto)
  } note p = this
  then have  $\forall i < \text{length } ts. ?p i$  by auto
} note main = this
{
  fix s
  assume steps:  $(\text{Fun } f \text{ ts} \cdot \sigma, s) \in (\text{nrqrstep nfs } Q R)^*$  and NF:  $\bigwedge u. u \triangleleft s \implies$ 
 $u \in \text{NF-terms } Q$ 
  from nrqrsteps-preserve-root[OF steps]
  obtain ss where s:  $s = \text{Fun } f \text{ ss}$  and len:  $\text{length } ss = \text{length } ts$  by (cases s,
auto)
  note main = main[OF len]
  {
    fix i
    assume i:  $i < \text{length } ts$ 
    from nrqrsteps-imp-arg-qrsteps[OF steps, of i] i len
    have steps:  $(ts ! i \cdot \sigma, ss ! i) \in \text{?QR}^*$  unfolding s by auto

```

```

    have NF:  $ss \vdash i \in NF\text{-terms } Q$ 
      by (rule NF, unfold s, insert i len, auto)
    note steps NF
  }
  note main = main[OF this]
  then have  $\tau: \bigwedge i. i < \text{length } ts \implies ?P \ ss \ i$  by blast
  have  $(\forall x \in \text{vars-term } (Fun \ f \ ts). (\sigma \ x, \tau \ x) \in ?QR^* \wedge (b \ x \longrightarrow \tau \ x \in NF\text{-terms } Q)) \wedge$ 
     $(Fun \ f \ ts \cdot \tau, s) \in ?QR^*$ 
  proof (intro conjI, intro ballI)
    fix x
    assume  $x \in \text{vars-term } (Fun \ f \ ts)$ 
    then obtain i where  $i: i < \text{length } ts$  and  $x: x \in \text{vars-term } (ts \vdash i)$ 
      by (auto simp: set-conv-nth)
    with  $\tau$  show  $(\sigma \ x, \tau \ x) \in ?QR^* \wedge (b \ x \longrightarrow \tau \ x \in NF\text{-terms } Q)$  by auto
  next
    from len have len:  $\text{length } (\text{map } (\lambda t. t \cdot \tau) \ ts) = \text{length } ss$  by simp
    show  $(Fun \ f \ ts \cdot \tau, s) \in ?QR^*$  unfolding s
      using  $\tau$ [THEN conjunct2]
      using args-grsteps-imp-grsteps[OF len, of nfs Q R f] by auto
    qed
  } note main = this
  from firstStep[OF qrstep-iff-rqrstep-or-nrqrstep Fun(2)]
  show ?case
  proof
    assume steps:  $(Fun \ f \ ts \cdot \sigma, s) \in (nrqrstep \ nfs \ Q \ R)^*$ 
    show ?thesis
    proof (rule main[OF steps])
      fix u
      assume sub:  $s \triangleright u$ 
      then have  $s \supseteq u$  by auto
      show  $u \in NF\text{-terms } Q$ 
        by (rule NF-subterm[OF Fun(3)], insert sub, auto)
      qed
    next
      assume steps:  $(Fun \ f \ ts \cdot \sigma, s) \in (nrqrstep \ nfs \ Q \ R)^* \ O \ rqrstep \ nfs \ Q \ R \ O$ 
      ?QR*
      then obtain u v where one:  $(Fun \ f \ ts \cdot \sigma, u) \in (nrqrstep \ nfs \ Q \ R)^*$ 
        and two:  $(u, v) \in rqrstep \ nfs \ Q \ R$ 
        and three:  $(v, s) \in ?QR^*$  by auto
      {
        fix u'
        assume  $u \triangleright u'$ 
        then have  $u' \in NF\text{-terms } Q$  using two[unfolded rqrstep-def] by auto
      }
      from main[OF one this]
      have first:  $\forall x \in \text{vars-term } (Fun \ f \ ts). (\sigma \ x, \tau \ x) \in ?QR^* \wedge (b \ x \longrightarrow \tau \ x \in$ 
         $NF\text{-terms } Q)$ 
        and steps:  $(Fun \ f \ ts \cdot \tau, u) \in ?QR^*$  by blast+

```

from *two* **have** $(u,v) \in ?QR$ **unfolding** *qrstep-iff-rqrstep-or-nrqrstep* **by** *auto*
with *three* **have** $steps2: (u,s) \in ?QR^*$ **by** *auto*
show *?thesis*
by (*intro conjI, rule first, insert steps steps2, auto*)
qed
qed

lemma *normalize-subst-qsteps-inn:*

fixes $Q R R' \sigma nfs$
defines $\tau: \tau \equiv \lambda x. some_NF (qrstep nfs Q R') (\sigma x)$
assumes *UNF: UNF (qrstep nfs Q R')*
and $R': \bigwedge x u. x \in vars_term t \implies (\sigma x, u) \in (qrstep nfs Q R)^* \implies (\sigma x, u) \in (qrstep nfs Q R')^*$
and $steps: (t \cdot \sigma, s) \in (qrstep nfs Q R)^*$
and $s: s \in NF_terms Q$
and $inn: NF_terms Q \subseteq NF_trs R'$
shows $(\forall x \in vars_term t. (\sigma x, \tau x) \in (qrstep nfs Q R')^* \wedge \tau x \in NF_terms Q) \wedge (t \cdot \tau, s) \in (qrstep nfs Q R)^*$
using *normalize-subst-qsteps-inn-partial[OF UNF R' steps s inn, of $\lambda -. True$, unfolded if-True]*
unfolding τ **by** *blast*

lemma *Tinf-imp-SN-nr-first-root-step-rel:*

assumes *Tinf: $t \in Tinf (relto (qrstep nfs Q R) (qrstep nfs Q' S))$*
shows $SN_on (relto (nrqrstep nfs Q R) (nrqrstep nfs Q' S)) \{t\} \wedge (\exists s u. (t,s) \in (nrqrstep nfs Q R \cup nrqrstep nfs Q' S)^* \wedge (s,u) \in rqrstep nfs Q R \cup rqrstep nfs Q' S \wedge \neg SN_on (relto (qrstep nfs Q R) (qrstep nfs Q' S)) \{u\})$
proof –
let $?R = qrstep nfs Q R$
let $?S = qrstep nfs Q' S$
let $?rel = relto ?R ?S$
let $?nR = nrqrstep nfs Q R$
let $?nS = nrqrstep nfs Q' S$
let $?nrel = relto ?nR ?nS$
let $?rR = rqrstep nfs Q R$
let $?rS = rqrstep nfs Q' S$
let $?N = ?nR \cup ?nS$
let $?RO = ?rR \cup ?rS$
from *Tinf[unfolded Tinf-def]* **have** $notSN: \neg SN_on ?rel \{t\}$ **and** $min: \bigwedge s. s \triangleleft t \implies SN_on ?rel \{s\}$ **by** *auto*
from *this[unfolded SN-rel-on-def[symmetric] SN-rel-on-conv]*
have $nSN: \neg SN_rel_on_alt ?R ?S \{t\}$ **and** $min2: \bigwedge s. s \triangleleft t \implies SN_rel_on_alt ?R ?S \{s\}$ **by** *auto*
from *nSN[unfolded SN-rel-on-alt-def]* **obtain** ts **where** $start: ts\ 0 = t$ **and** $steps: \bigwedge i. (ts\ i, ts\ (Suc\ i)) \in ?R \cup ?S$ **and** $inf: INFM\ i. (ts\ i, ts\ (Suc\ i)) \in ?R$ **by** *auto*
show *?thesis*
proof (*cases SN-on ?nrel {t}*)
case *True*
from *True[unfolded SN-rel-on-def[symmetric] SN-rel-on-conv]*

```

have SN-rel-on-alt ?nR ?nS {ts 0} unfolding start .
note SN = this[unfolded SN-rel-on-alt-def, rule-format, of ts]
let ?P =  $\lambda i. (ts\ i, ts\ (Suc\ i)) \in ?RO$ 
have  $\exists i. ?P\ i$ 
proof (rule ccontr)
  assume nthesis:  $\neg ?thesis$ 
  then have  $\bigwedge i. (ts\ i, ts\ (Suc\ i)) \in ?N$  using steps[unfolded qrstep-iff-rqrstep-or-nrqrstep]
by auto
  with SN have  $\neg (INFM\ i. (ts\ i, ts\ (Suc\ i)) \in ?nR)$  by auto
  then show False
  proof (elim notE, unfold INFM-nat-le, intro allI)
    fix m
    from inf[unfolded INFM-nat-le] obtain n where  $n: n \geq m$  and step:  $(ts\ n, ts\ (Suc\ n)) \in ?R$  by auto
    from step nthesis have  $(ts\ n, ts\ (Suc\ n)) \in ?nR$  unfolding qrstep-iff-rqrstep-or-nrqrstep
  by auto
  then show  $\exists n \geq m. (ts\ n, ts\ (Suc\ n)) \in ?nR$  using n by auto
  qed
qed
then obtain i where step: ?P i by auto
from LeastI[of ?P, OF step] have step: ?P (LEAST i. ?P i) .
obtain i where i: i = (LEAST i. ?P i) by auto
from step have step: ?P i unfolding i .
{
  fix j
  assume j < i
  from not-less-Least[OF this[unfolded i]] have  $\neg ?P\ j$  .
  with steps[of j] have  $(ts\ j, ts\ (Suc\ j)) \in ?N$ 
  unfolding qrstep-iff-rqrstep-or-nrqrstep by auto
} note nsteps = this
show ?thesis
proof -
  have  $(ts\ 0, ts\ i) \in ?N^*$  unfolding rtrancl-fun-conv
  by (rule exI[of - ts], rule exI[of - i], insert nsteps, auto)
  moreover
  let ?ss = shift ts (Suc i)
  have  $\neg SN\text{-on}\ (relto\ ?R\ ?S)\ \{ts\ (Suc\ i)\}$ 
  unfolding SN-rel-on-def[symmetric]
  unfolding SN-rel-on-conv SN-rel-on-alt-def
  unfolding not-all not-imp not-not
  proof (rule exI[of - ?ss], intro conjI allI)
    fix j
    show  $(?ss\ j, ?ss\ (Suc\ j)) \in ?R \cup ?S$  using steps[of j + Suc i] by auto
  next
  show ?ss 0  $\in \{ts\ (Suc\ i)\}$  by simp
  next
  show INFM j.  $(?ss\ j, ?ss\ (Suc\ j)) \in ?R$ 
  unfolding INFM-nat-le
  proof (rule allI)

```

```

    fix m
    from inf[unfolded INFM-nat-le, rule-format, of m + Suc i]
    obtain n where n: n ≥ m + Suc i and step: (ts n, ts (Suc n)) ∈ ?R by
auto
    show ∃ n ≥ m. (?ss n, ?ss (Suc n)) ∈ ?R
    by (rule exI[of - n - Suc i], insert n step, auto)
  qed
  qed
  ultimately show ?thesis
  using True step start by blast
  qed
next
case False
from this[unfolded SN-rel-on-def[symmetric] SN-rel-on-conv]
have nSN: ¬ SN-rel-on-alt ?nR ?nS {t} .
  from nSN[unfolded SN-rel-on-alt-def] obtain ts where start: ts 0 = t and
nsteps: ∧ i. (ts i, ts (Suc i)) ∈ ?N and inf: INFM i. (ts i, ts (Suc i)) ∈ ?nR by
auto
  obtain f ss where init: ts 0 = Fun f ss
  using nrqrstep-imp-arg-grstep[of ts 0 ts (Suc 0)] nsteps[of 0]
  by (cases ts 0, auto)
  let ?n = length ss
  obtain n where n: n = ?n by auto
  {
    fix i
    have ∃ ssi. ts i = Fun f ssi ∧ length ssi = n
    proof (induct i)
      case 0
      then show ?case unfolding init n by auto
    next
      case (Suc i)
      from Suc obtain ssi where tsi: ts i = Fun f ssi and n: length ssi = n by
auto
      from nrqrstep-preserves-root[of ts i ts (Suc i)] nsteps[of i]
      have root (ts (Suc i)) = root (ts i) by auto
      then show ?case unfolding tsi root.simps n by (cases ts (Suc i), auto)
    qed
  }
  from choice[OF allI[OF this]] obtain sss where ts: ∧ i. ts i = Fun f (sss i)
and len: ∧ i. length (sss i) = n by force
  let ?strict = λ i. ∃ j < n. (sss i ! j, sss (Suc i) ! j) ∈ ?R
  have inf: INFM i. ?strict i
  unfolding INFM-nat-le
  proof (intro allI)
    fix i
    from inf[unfolded INFM-nat-le, rule-format, of i] obtain k where
    k: k ≥ i and step: (ts k, ts (Suc k)) ∈ ?nR by auto
    show ∃ k ≥ i. ?strict k
    by (rule exI, rule conjI[OF k],

```

```

      insert nrqrstep-imp-arg-qrstep[OF step] ts len, auto)
qed
let ?idx =  $\lambda i.$  if ?strict i then (SOME j. j < n  $\wedge$  (sss i ! j, sss (Suc i) ! j)  $\in$ 
?R)
  else 0
obtain idx where idx: idx = ?idx by auto
{
  fix i
  assume stri: ?strict i
  then have idxi: idx i = (SOME j. j < n  $\wedge$  (sss i ! j, sss (Suc i) ! j)  $\in$  ?R)
unfolding idx by simp
  from someI-ex[OF stri] have idx i < n  $\wedge$  (sss i ! idx i, sss (Suc i) ! idx i)  $\in$ 
?R unfolding idxi .
} note idx-strict = this
{
  fix i
  have idx i  $\leq$  n using idx-strict[of i] by (cases ?strict i, auto simp: idx)
} note idx-n = this
{
  fix X
  have finite (idx ' X)
  proof (rule finite-subset)
    show idx ' X  $\subseteq$  { i . i  $\leq$  n } using idx-n by auto
  next
    have id: {i. i  $\leq$  n} = set [0 ..< Suc n] by auto
    show finite {i. i  $\leq$  n} unfolding id by (rule finite-set)
  qed
} note fin = this
note pigeon = pigeonhole-infinite[OF inf[unfolded INFM-iff-infinite] fin]
then obtain j where infinite { i  $\in$  {i. ?strict i}. idx i = j } by auto
then have inf: INFM i. ?strict i  $\wedge$  idx i = j unfolding INFM-iff-infinite by
simp
note inf = inf[unfolded INFM-nat-le, rule-format]
{
  from inf[of 0] obtain k where stri: ?strict k and idx: idx k = j by auto
  from idx-strict[OF stri] idx have j < n by auto
} note jn = this
with n have ss ! j  $\in$  set ss by auto
with init have ss ! j  $\triangleleft$  ts 0 by auto
from min[unfolded start[symmetric], OF this] have SN: SN-on ?rel {ss ! j} by
auto
let ?ss =  $\lambda i.$  sss i ! j
let ?RS = ?R  $\cup$  ?S
from SN have SN: SN-on ?rel {?ss 0} using ts[of 0] init by simp
{
  fix i
  have (?ss i, ?ss (Suc i))  $\in$  ?RS $^{\wedge} =$ 
  proof (cases (ts i, ts (Suc i))  $\in$  ?nS)
    case True

```

```

    from nrqrstep-imp-arg-grsteps[OF True, of j]
    show ?thesis unfolding ts by simp
  next
    case False
    with nsteps[of i]
    have (ts i, ts (Suc i)) ∈ ?nR by auto
    from nrqrstep-imp-arg-grsteps[OF this, of j]
    show ?thesis unfolding ts by auto
  qed
} note j-steps = this
let ?Rel = ?RS* O ?R O ?RS*
{
  fix i
  have (?ss i, ?ss (Suc i)) ∈ ?RS* ∪ ?Rel
    by (rule set-mp[OF - j-steps[of i]], regexp)
} note jsteps = this
from SN-on-trancl[OF SN, unfolded relto-trancl-conv]
have SN: SN-on ?Rel {?ss 0} .
have compat: ?RS* O ?Rel ⊆ ?Rel by regexp
from non-strict-ending[of ?ss ?RS* ?Rel, OF allI[OF jsteps] compat SN]
obtain k where k:  $\bigwedge l. l \geq k \implies (?ss l, ?ss (Suc l)) \notin ?Rel$  by auto
from inf[of k] obtain l where lk:  $l \geq k$  and strict: ?strict l and lj: idx l = j
by auto
from k[OF lk] have no-step: (?ss l, ?ss (Suc l))  $\notin$  ?R by auto
with idx-strict[OF strict] lj have False by auto
then show ?thesis by auto
qed
qed

lemma nrqrstep-empty[simp]: nrqrstep nfs Q {} = {} unfolding nrqrstep-def by
auto

lemma Tinf-imp-SN-nr-first-root-step:
  assumes Tinf:  $t \in Tinf (qrstep\ nfs\ Q\ R)$ 
  shows SN-on (nrqrstep nfs Q R) {t}  $\wedge (\exists s\ u. (t,s) \in (nrqrstep\ nfs\ Q\ R)^* \wedge (s,u) \in qrstep\ nfs\ Q\ R \wedge \neg SN-on (qrstep\ nfs\ Q\ R) \{u\})$ 
  using Tinf-imp-SN-nr-first-root-step-rel[of t nfs Q {} Q R]
  using Tinf by auto

lemma SN-on-subterms-imp-SN-on-nrqrstep:
  fixes R :: ('f, 'v) trs
  assumes  $\forall s \triangleleft t. SN-on (qrstep\ nfs\ Q\ R) \{s\}$ 
  shows SN-on (nrqrstep nfs Q R) {t}
proof (cases SN-on (qrstep nfs Q R) {t})
  case True
  then show ?thesis unfolding qrstep-iff-rqrstep-or-nrqrstep unfolding SN-defs
  by auto
next
  case False

```

with *assms* **have** $t \in \text{Tinf} \text{ (qrstep nfs } Q \text{ } R)$ **unfolding** *Tinf-def* **by** *auto*
from *Tinf-imp-SN-nr-first-root-step*[*OF this*]
show *?thesis* **by** *simp*
qed

lemma *SN-args-imp-SN-rel*:

assumes *SN*: $\bigwedge s. s \in \text{set } ss \implies \text{SN-on } (\text{relto } (\text{qrstep nfs } Q \text{ } R) \text{ (qrstep nfs } Q \text{ } S)) \{s\}$
and *nvar*: $\forall (l, r) \in R \cup S. \text{is-Fun } l$
and *ndef*: $\neg \text{defined } (\text{applicable-rules } Q \text{ (} R \cup S)) (f, \text{length } ss)$
shows *SN-on* $(\text{relto } (\text{qrstep nfs } Q \text{ } R) \text{ (qrstep nfs } Q \text{ } S)) \{\text{Fun } f \text{ } ss\}$
proof (*rule ccontr*)
assume *nSN*: $\neg ?thesis$
let *?RS* = $\text{relto } (\text{qrstep nfs } Q \text{ } R) \text{ (qrstep nfs } Q \text{ } S)$
have $\text{Fun } f \text{ } ss \in \text{Tinf } ?RS$ **unfolding** *Tinf-def*
proof (*intro CollectI conjI nSN allI impI*)
fix *s*
assume $\text{Fun } f \text{ } ss \triangleright s$
then obtain *si* **where** *si*: $si \in \text{set } ss$ **and** *supteq*: $si \triangleright s$ **by** *auto*
from *ctxt-closed-SN-on-subt*[*OF ctxt.closed-relto*[*OF ctxt-closed-qrstep ctxt-closed-qrstep*]
SN[*OF si*] *supteq*]
show *SN-on* $?RS \{s\}$.
qed
from *Tinf-imp-SN-nr-first-root-step-rel*[*OF this*]
obtain *s u* **where** *steps*: $(\text{Fun } f \text{ } ss, s) \in (\text{nrqrstep nfs } Q \text{ (} R \cup S))^{*}$ **and** *su*: $(s, u) \in \text{qrstep nfs } Q \text{ (} R \cup S)$
unfolding *nrqrstep-union qrstep-union* **by** *auto*
from *nrqrsteps-preserve-root*[*OF steps*] **obtain** *ts* **where**
s: $s = \text{Fun } f \text{ } ts$ **and**
len: $\text{length } ss = \text{length } ts$ **by** (*cases s, auto*)
note *ndef* = *ndef*[*unfolded len*]
note *su* = *su*[*unfolded s*]
from *qrstepE*[*OF su*] **obtain** *l r* σ **where** *lr*: $(l, r) \in (R \cup S)$ **and** *id*: $\text{Fun } f \text{ } ts = l \cdot \sigma$
and *NF*: $\forall u. u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$.
from *only-applicable-rules*[*OF NF*] **have** *app*: *applicable-rule* $Q (l, r)$.
from *nvar lr id* **obtain** *ls* **where** *l*: $l = \text{Fun } f \text{ } ls$ **and** *len*: $\text{length } ts = \text{length } ls$
by (*cases l; force*)
note *ndef* = *ndef*[*unfolded len*]
from *app lr ndef* **show** *False* **unfolding** *l* **unfolding** *applicable-rules-def defined-def* **by** *force*
qed

lemma *SN-args-imp-SN*:

assumes $\bigwedge s. s \in \text{set } ss \implies \text{SN-on } (\text{qrstep nfs } Q \text{ } R) \{s\}$
and $\forall (l, r) \in R. \text{is-Fun } l$
and $\neg \text{defined } (\text{applicable-rules } Q \text{ } R) (f, \text{length } ss)$
shows *SN-on* $(\text{qrstep nfs } Q \text{ } R) \{\text{Fun } f \text{ } ss\}$

```

using SN-args-imp-SN-rel[of ss nfs Q {} R f] assms by auto

lemma SN-args-imp-SN-rel-rstep:
  assumes  $\bigwedge s. s \in \text{set } ss \implies \text{SN-on } (\text{relto } (\text{rstep } R) (\text{rstep } S)) \{s\}$ 
  and  $\forall (l, r) \in R \cup S. \text{is-Fun } l$ 
  and  $\neg \text{defined } (R \cup S) (f, \text{length } ss)$ 
  shows  $\text{SN-on } (\text{relto } (\text{rstep } R) (\text{rstep } S)) \{\text{Fun } f \text{ } ss\}$ 
  using SN-args-imp-SN-rel[of ss False {} S R f] assms by auto

lemma SN-args-imp-SN-rstep :
  assumes SN:  $\bigwedge s. s \in \text{set } ss \implies \text{SN-on } (\text{rstep } R) \{s\}$ 
  and nvar:  $\forall (l, r) \in R. \text{is-Fun } l$ 
  and ndef:  $\neg \text{defined } R (f, \text{length } ss)$ 
  shows  $\text{SN-on } (\text{rstep } R) \{\text{Fun } f \text{ } ss\}$ 
  using SN-args-imp-SN[of ss False {} - f, OF - nvar] SN ndef by auto

lemma normalize-subst-qrstps-inn-infinite:
  fixes Q R R' σ nfs
  defines  $\tau: \tau \equiv \lambda x. \text{some-NF } (\text{qrstep } \text{nfs } Q \text{ } R') (\sigma \text{ } x)$ 
  assumes UNF:  $\text{UNF } (\text{qrstep } \text{nfs } Q \text{ } R')$ 
  and R':  $\bigwedge x \text{ } u. x \in \text{vars-term } t \implies (\sigma \text{ } x, u) \in (\text{qrstep } \text{nfs } Q \text{ } R)^* \implies (\sigma \text{ } x, u) \in (\text{qrstep } \text{nfs } Q \text{ } R')^*$ 
  and steps:  $\neg \text{SN-on } (\text{qrstep } \text{nfs } Q \text{ } R) \{t \cdot \sigma\}$ 
  and SN:  $\bigwedge x. x \in \text{vars-term } t \implies \text{SN-on } (\text{qrstep } \text{nfs } Q \text{ } R) \{\sigma \text{ } x\}$ 
  and inn:  $\text{NF-terms } Q \subseteq \text{NF-trs } R'$ 
  shows  $\neg \text{SN-on } (\text{qrstep } \text{nfs } Q \text{ } R) \{t \cdot \tau\}$ 
proof –
  let ?R = qrstep nfs Q R
  let ?R' = qrstep nfs Q R'
  from not-SN-imp-subt-Tinf[OF steps] obtain s
    where subt:  $t \cdot \sigma \sqsupseteq s$  and s-inf:  $s \in \text{Tinf } ?R$  by auto
  from supteq-imp-subt-at[OF subt] obtain p where p:  $p \in \text{poss } (t \cdot \sigma)$ 
    and subt:  $s = t \cdot \sigma \mid\!-\! p$  by auto
  show ?thesis
  proof (cases p ∈ poss t ∧ is-Fun (t | - p))
    case True
      obtain tp where tp:  $tp = t \mid\!-\! p$  by auto
      with True have p:  $p \in \text{poss } t$  and is-fun:  $\text{is-Fun } tp$  by auto
      then have s:  $s = tp \cdot \sigma$  by (simp add: subt tp)
      from subt-at-imp-supteq[OF p] have subt:  $t \sqsupseteq tp$  unfolding tp .
      from s-inf[unfolded s] have tp-inf:  $tp \cdot \sigma \in \text{Tinf } ?R$  .
      {
        fix x
        assume  $x \in \text{vars-term } tp$ 
        then have  $x \in \text{vars-term } t$  using supteq-imp-vars-term-subset [OF subt] by
auto
      }
      note vars = this
      let ?n = nrqrstep nfs Q R
      let ?r = rqrstep nfs Q R

```

```

from  $tp\text{-}inf[unfolding\ Tinf\text{-}def]$  have  $SN: \forall s \triangleleft tp \cdot \sigma. SN\text{-}on\ ?R\ \{s\}$ 
and  $nSN: \neg SN\text{-}on\ ?R\ \{tp \cdot \sigma\}$  by auto
from  $SN\text{-}on\text{-}subterms\text{-}imp\text{-}SN\text{-}on\text{-}nrqrstep[OF\ SN]$  have  $SN: SN\text{-}on\ ?n\ \{tp \cdot$ 
 $\sigma\}$ 
by auto
from  $nSN$  obtain  $f$  where  $start: f\ 0 = tp \cdot \sigma$ 
and  $steps: \forall i. (f\ i, f\ (Suc\ i)) \in ?R$  by auto
from  $chain\text{-}Un\text{-}SN\text{-}on\text{-}imp\text{-}first\text{-}step[OF\ steps[unfolding\ qrstep\text{-}iff\text{-}rqrstep\text{-}or\text{-}nrqrstep],$ 
 $OF\ SN[unfolding\ start[symmetric]]]$ 
obtain  $i$  where  $root: (f\ i, f\ (Suc\ i)) \in ?r$  and  $nroot: \bigwedge j. j < i \implies (f\ j, f$ 
 $(Suc\ j)) \in ?n$  by auto
have  $nroot: (tp \cdot \sigma, f\ i) \in ?n^*$  unfolding  $rtrancl\text{-}fun\text{-}conv$ 
by  $(intro\ exI[of\ -\ f]\ exI[of\ -\ i],\ unfold\ start,\ insert\ nroot,\ auto)$ 
from  $is\text{-}fun$  obtain  $g\ ts$  where  $is\text{-}fun: tp = Fun\ g\ ts$  by  $(cases\ tp,\ auto)$ 
note  $nroot = nroot[unfolding\ is\text{-}fun]$ 
from  $nrqrsteps\text{-}preserve\text{-}root[OF\ nroot]$  obtain  $ss$  where  $fi: f\ i = Fun\ g\ ss$  and
 $len: length\ ss = length\ ts$  by  $(cases\ f\ i,\ auto)$ 
note  $root = root[unfolding\ fi]$ 
from  $root[unfolding\ rqrstep\text{-}def]$  have  $NF: \bigwedge u. u \triangleleft Fun\ g\ ss \implies u \in NF\text{-}terms$ 
 $Q$  by auto
let  $?ts = map\ (\lambda\ t. t \cdot \tau)\ ts$ 
let  $?lts = length\ ?ts$ 
{
fix  $i$ 
assume  $i: i < ?lts$ 
from  $nrqrsteps\text{-}imp\text{-}arg\text{-}qrsteps[OF\ nroot,\ of\ i]\ i\ len$ 
have  $steps: (ts\ !\ i \cdot \sigma, ss\ !\ i) \in ?R^*$  unfolding  $fi$  by auto
have  $NF: ss\ !\ i \in NF\text{-}terms\ Q$ 
by  $(rule\ NF,\ insert\ i\ len,\ auto)$ 
{
fix  $x$ 
assume  $x \in vars\text{-}term\ (ts\ !\ i)$ 
with  $i$  have  $x \in vars\text{-}term\ (Fun\ g\ ts)$  by auto
with  $vars$  have  $x \in vars\text{-}term\ t$  unfolding  $is\text{-}fun$  by auto
} note  $vars = this$ 
from  $normalize\text{-}subst\text{-}qrsteps\text{-}inn[OF\ UNF\ R'[OF\ vars]\ steps\ NF\ inn]$ 
have  $(ts\ !\ i \cdot \tau, ss\ !\ i) \in ?R^*$  unfolding  $\tau\ ..$ 
then have  $(?ts\ !\ i, ss\ !\ i) \in ?R^*$  using  $i$  by auto
} note  $\tau steps = this$ 
from  $len$  have  $len: ?lts = length\ ss$  by simp
from  $args\text{-}qrsteps\text{-}imp\text{-}qrsteps[OF\ len,\ of\ nfs\ Q\ R\ g]\ \tau steps$ 
have  $tpfi: (tp \cdot \tau, f\ i) \in ?R^*$  unfolding  $is\text{-}fun\ fi$  by simp
obtain  $g$  where  $g: g \equiv \lambda j. f\ (i + j)$  by auto
from  $steps$  have  $\bigwedge i. (g\ i, g\ (Suc\ i)) \in ?R$  unfolding  $g$  by auto
then have  $\neg SN\text{-}on\ ?R\ \{g\ 0\}$  by auto
then have  $nSN: \neg SN\text{-}on\ ?R\ \{f\ i\}$  unfolding  $g$  by auto
with  $steps\text{-}preserve\text{-}SN\text{-}on[OF\ tpfi]$ 
have  $\neg SN\text{-}on\ ?R\ \{tp \cdot \tau\}$  by auto
with  $ctxt\text{-}closed\text{-}SN\text{-}on\text{-}subt[OF\ ctxt\text{-}closed\text{-}qrstep\text{-}sup\text{-}teq\text{-}subst[OF\ subt],\ of$ 

```

```

nfs Q R  $\tau$ ]
  show ?thesis by auto
next
  case False
  from pos-into-subst[OF refl p False]
  obtain q q' x where pq: p = q @ q' and q: q  $\in$  poss t and t: t  $\vdash$  q = Var x
by auto
  from subteq-Var-imp-in-vars-term [OF sub-at-imp-supteq [OF q, unfolded t]]
  have x: x  $\in$  vars-term t .
  have s =  $\sigma$  x  $\vdash$  q'  $\wedge$  q'  $\in$  poss ( $\sigma$  x)
  using p
  unfolding poss-append-poss sub t pq
  and sub-at-append [OF poss-imp-subst-poss [OF q]]
  and sub-at-subst [OF q] t by simp
  then have s:  $\sigma$  x  $\supseteq$  s using sub-at-imp-supteq by auto
  from ctxt-closed-SN-on-subst[OF ctxt-closed-qstep SN[OF x] s]
  have SN-on ?R {s} by auto
  with s-inf[unfolded Tinf-def] show ?thesis by auto
qed
qed

```

lemma qstep-r-p-s-conv:

```

  fixes s t
  assumes step: (s, t)  $\in$  qstep-r-p-s nfs Q R lr p  $\sigma$ 
  defines [simp]: C  $\equiv$  ctxt-of-pos-term p s
  shows p  $\in$  poss s and p  $\in$  poss t and s = C⟨fst lr  $\cdot$   $\sigma$ ⟩ and t = C⟨snd lr  $\cdot$   $\sigma$ ⟩
and NF-subst nfs lr  $\sigma$  Q and ctxt-of-pos-term p t = C
proof -
  from step have p-in-s: p  $\in$  poss s unfolding qstep-r-p-s-def by simp
  have p-in-t: p  $\in$  poss t using qstep-r-p-s-imp-poss[OF step] by simp
  have C: C = ctxt-of-pos-term p t using ctxt-of-pos-term-qstep-below[OF step]
by simp

```

```

  from p-in-s show p  $\in$  poss s.
  show p  $\in$  poss t using qstep-r-p-s-imp-poss[OF step] by simp
  from step show s = C⟨fst lr  $\cdot$   $\sigma$ ⟩ unfolding qstep-r-p-s-def using ctxt-supt-id[OF
p-in-s] by simp
  from C show t = C⟨snd lr  $\cdot$   $\sigma$ ⟩ using step unfolding qstep-r-p-s-def using
ctxt-supt-id[OF p-in-t] by simp
  from step show NF-subst nfs lr  $\sigma$  Q unfolding qstep-r-p-s-def by simp
  from C show ctxt-of-pos-term p t = C by simp
qed

```

lemma qstep-r-p-s-redex-reduct:

```

  assumes step: (s,t)  $\in$  qstep-r-p-s nfs Q R lr p  $\sigma$ 
  shows t  $\vdash$  p = snd lr  $\cdot$   $\sigma$ 
  using qstep-r-p-s-conv[OF step]
  by (metis replace-at-subst-at)

```

lemma *qrstep-r-p-s-imp-nrqrstep*:
assumes *step*: $(s, t) \in \text{qrstep-r-p-s nfs } Q \ R \ \text{lr } p \ \sigma$ **and** *ne*: $p \neq []$
shows $(s, t) \in \text{nrqrstep nfs } Q \ R$
proof –
from *step*[*unfolded qrstep-r-p-s-def*]
have *NF*: $\forall u. u \triangleleft \text{fst } \text{lr} \cdot \sigma. u \in \text{NF-terms } Q$
and *p*: $p \in \text{poss } s$
and *lr*: $\text{lr} \in R$
and *s*: $s \mid\!-\ p = \text{fst } \text{lr} \cdot \sigma$
and *t*: $t = \text{replace-at } s \ p \ (\text{snd } \text{lr} \cdot \sigma)$
and *nfs*: *NF*-subst *nfs* *lr* $\sigma \ Q$ **by** *auto*
show *?thesis*
proof (*rule nrqrstepI*[*OF NF - - t*])
show $(\text{fst } \text{lr}, \text{snd } \text{lr}) \in R$ **using** *lr* **by** *simp*
next
from *ne* **obtain** *i q* **where** *ne*: $p = (i \# q)$ **by** (*cases p, auto*)
with *p* **show** *ctxt-of-pos-term* *p s* $\neq []$ **by** (*cases s, auto*)
qed (*insert ctxt-supt-id*[*OF p, unfolded s*] *nfs, auto*)
qed

lemma *qrsteps-imp-qrsteps-r-p-s*:
assumes $(s, t) \in (\text{qrstep nfs } Q \ R)^*$
shows $\exists n \ \text{ts} \ \text{lr} \ p \ \sigma. \ \text{ts } 0 = s \wedge \text{ts } n = t \wedge (\forall i < n. (\text{ts } i, \text{ts } (\text{Suc } i)) \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ (p \ i) \ (\sigma \ i))$
proof –
from *assms*[*unfolded qrsteps-rules-conv*]
obtain *n ts lr* **where**
first: $\text{ts } 0 = s$
and *last*: $\text{ts } n = t$
and *steps*: $\forall i < n. (\text{ts } i, \text{ts } (\text{Suc } i)) \in \text{qrstep nfs } Q \ \{\text{lr } i\} \wedge \text{lr } i \in R$ **by** *auto*
let *?ts* $= \lambda i. (\text{ts } i, \text{ts } (\text{Suc } i))$
{
fix *i*
assume $i < n$
from *steps*[*rule-format, OF this*]
have $?ts \ i \in \text{qrstep nfs } Q \ (R \cap \{\text{lr } i\})$ **by** *auto*
from *this*[*unfolded qrstep-qrstep-r-p-s-conv*] **obtain** *lr' p* σ **where** $?ts \ i \in \text{qrstep-r-p-s nfs } Q \ (R \cap \{\text{lr } i\}) \ \text{lr}' \ p \ \sigma$ **by** *auto*
then **have** $\exists p \ \sigma. ?ts \ i \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ p \ \sigma$ **unfolding** *qrstep-r-p-s-def*
by *auto*
}
then **have** $\forall i. \exists p \ \sigma. i < n \longrightarrow ?ts \ i \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ p \ \sigma$ **by** *simp*
from *choice*[*OF this*] **obtain** *p* **where** $\forall i. \exists \sigma. i < n \longrightarrow ?ts \ i \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ (p \ i) \ \sigma$ **by** *auto*
from *choice*[*OF this*] **obtain** σ **where** $\bigwedge i. i < n \implies ?ts \ i \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ (p \ i) \ (\sigma \ i)$ **by** *auto*
show *?thesis*
proof (*intro exI conjI*)
show $\forall i < n. ?ts \ i \in \text{qrstep-r-p-s nfs } Q \ R \ (\text{lr } i) \ (p \ i) \ (\sigma \ i)$ **using** *steps* **by**

simp
qed (*insert first last, auto*)
qed

lemma *qrsteps-r-p-s-imp-qrsteps*:
assumes *first*: $ts\ 0 = s$
and *last*: $ts\ n = t$
and *steps*: $\bigwedge i. i < n \implies (ts\ i, ts\ (Suc\ i)) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ (lr\ i)\ (p\ i)$
 $(\sigma\ i)$
shows $(s, t) \in (qrstep\ nfs\ Q\ R)^*$
unfolding *qrsteps-rules-conv*
proof (*intro exI[of - n] exI[of - ts] exI[of - lr] conjI first last allI, intro allI impI*)
fix *i*
assume *i*: $i < n$
show $(ts\ i, ts\ (Suc\ i)) \in qrstep\ nfs\ Q\ \{lr\ i\} \wedge lr\ i \in R$ **using** *steps[OF i]*
unfolding *qrstep-qrstep-r-p-s-conv*
unfolding *qrstep-r-p-s-def* **by** *blast*
qed

lemma *qrstep-r-p-s-imp-applicable-rule*: $(s, t) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ lr\ p\ \sigma \implies$
applicable-rule $Q\ lr$
using *only-applicable-rules[of fst lr σ Q]*
unfolding *qrstep-r-p-s-def*
by (*cases lr, simp*)

lemma *SN-on-qrstep-r-p-s-imp-wwf-rule*:
assumes *SN*: *SN-on* $(qrstep\ nfs\ Q\ R)\ \{t\}$
and *step*: $(t, s) \in qrstep\text{-}r\text{-}p\text{-}s\ nfs\ Q\ R\ lr\ p\ \sigma$
and *nfs*: $\neg nfs$
shows $vars\text{-}term\ (snd\ lr) \subseteq vars\text{-}term\ (fst\ lr) \wedge is\text{-}Fun\ (fst\ lr)$
proof –
obtain *l r* **where** $lr = (l, r)$ **by** *force*
from *qrstep-r-p-s-imp-applicable-rule[OF step]* **have** *u*: *applicable-rule* $Q\ lr$.
from *step[unfolded lr qrstep-r-p-s-def]* **have** *p*: $p \in poss\ t$ **and** *t*: $t \mid\!-\ p = l \cdot \sigma$
and *mem*: $(l, r) \in R$ **and** *NF*: $\forall u \triangleleft l \cdot \sigma. u \in NF\text{-}terms\ Q$ **by** *auto*
from *SN-on-imp-wwf-rule[OF SN ctxt-supt-id[OF p, unfolded t, symmetric] mem NF nfs]*
have *wwf-rule* $Q\ (l, r)$.
then show *?thesis* **using** *u* **unfolding** *lr wwf-rule-def* **by** *auto*
qed

lemma *SN-on-Var-gen*:
assumes *Ball* $(fst\ 'R)\ is\text{-}Fun$ **shows** *SN-on* $(qrstep\ nfs\ Q\ R)\ \{Var\ x\}$ (*is* *SN-on* $\{?x\}$)
proof –
let *?qr* = *qrstep nfs Q R*
show *SN-on* *?qr* $\{?x\}$
proof (*rule ccontr*)
assume $\neg ?thesis$

```

then obtain  $A$  where  $A \ 0 = ?x$  and  $rsteps: chain \ ?qr \ A$ 
  unfolding  $SN-on-def$  by best
then have  $(?x, A \ 1) \in ?qr$  using  $spec[OF \ rsteps, of \ 0]$  by auto
then obtain  $l \ r \ C \ \sigma$  where  $(l, r) \in R$  and  $x: Var \ x = C \langle l \cdot \sigma \rangle$  by auto
with  $assms$  obtain  $f \ ls$  where  $l = Fun \ f \ ls$  by force
with  $x$  obtain  $ls$  where  $Var \ x = C \langle Fun \ f \ ls \rangle$  by auto
then show  $False$  by (cases  $C$ , auto)
qed
qed

lemma  $SN-on-Var$ :
  assumes  $wwf-qtrs \ Q \ R$  shows  $SN-on \ (qrstep \ nfs \ Q \ R) \ \{Var \ x\}$  (is  $SN-on - \ \{?x\}$ )
  by (rule  $SN-on-Var-gen$ , insert  $wwf-qtrs-imp-left-fun[OF \ assms]$ , force)

lemma  $wwf-qtrs-imp-nfs-switch-r-p-s$ : assumes  $wwf: wwf-qtrs \ Q \ R$ 
  shows  $qrstep-r-p-s \ nfs \ Q \ R = qrstep-r-p-s \ nfs' \ Q \ R$ 
proof -
  {
    fix  $nfs \ nfs' \ lr \ p \ \sigma \ s \ t$ 
    assume  $step: (s, t) \in qrstep-r-p-s \ nfs \ Q \ R \ lr \ p \ \sigma$ 
    obtain  $l \ r$  where  $lr: lr = (l, r)$  by force
    from  $step[unfolded \ qrstep-r-p-s-def \ lr]$  have *:  $(\forall u \triangleleft l \cdot \sigma. u \in NF-terms \ Q)$ 
       $p \in poss \ s \ (l, r) \in R \ s \vdash p = l \cdot \sigma \ t = (ctxt-of-pos-term \ p \ s) \langle r \cdot \sigma \rangle$  and  $NF:$ 
 $NF-subst \ nfs \ (l, r) \ \sigma \ Q$  by auto
    have  $applicable-rule \ Q \ (l, r)$  using  $only-applicable-rules \ *$  by auto
    with  $wwf[unfolded \ wwf-qtrs-def] \ *$  have  $l: is-Fun \ l$  and  $rl: vars-term \ r \subseteq$ 
 $vars-term \ l$  by auto
    have  $NF: NF-subst \ nfs' \ (l, r) \ \sigma \ Q$ 
      unfolding  $NF-subst-def$ 
    proof (intro  $impI \ subsetI$ )
      fix  $u$ 
      assume  $u: u \in \sigma \text{ 'vars-rule } (l, r)$ 
      then obtain  $x$  where  $u: u = \sigma \ x$  and  $x: x \in vars-rule \ (l, r)$  by auto
      from  $x \ rl$  have  $x: x \in vars-term \ l$  by (cases  $x \in vars-term \ l$ , auto simp:
 $vars-rule-def$ )
      then have  $Var \ x \trianglelefteq l$  by auto
      with  $l$  have  $Var \ x \triangleleft l$  by auto
      from  $supt-subst[OF \ this, of \ \sigma] \ *$ 
      show  $u \in NF-terms \ Q$  unfolding  $u$  by auto
    qed
    with * have  $(s, t) \in qrstep-r-p-s \ nfs' \ Q \ R \ lr \ p \ \sigma$  unfolding  $lr \ qrstep-r-p-s-def$ 
  by auto
  } note  $main = this$ 
  from  $main[of \ - \ - \ nfs \ - \ - \ nfs'] \ main[of \ - \ - \ nfs' \ - \ - \ nfs]$  show  $?thesis$ 
  by (intro  $ext$ , auto)
qed

```

```

lemma  $wwf-qtrs-imp-nfs-switch$ : assumes  $wwf: wwf-qtrs \ Q \ R$ 
  shows  $qrstep \ nfs \ Q \ R = qrstep \ nfs' \ Q \ R$ 

```

```

using
  qrstep-qrstep-r-p-s-conv[of - - nfs Q R]
  qrstep-qrstep-r-p-s-conv[of - - nfs' Q R]
  wwf-qtrs-imp-nfs-switch-r-p-s[OF wwf, of nfs nfs'] by auto

lemma wwf-qtrs-imp-nfs-False-switch: assumes nfs  $\implies$  Q  $\neq$  {}  $\implies$  wwf-qtrs Q
R
  shows qrstep nfs Q R = qrstep False Q R
proof (cases nfs  $\wedge$  Q  $\neq$  {})
  case True
    from wwf-qtrs-imp-nfs-switch[OF assms] True show ?thesis by auto
  next
    case False
      show ?thesis
      proof (cases nfs)
        case True
          with False have Q = {} by auto
          then show ?thesis by simp
        case False
          qed simp
      qed
    qed

lemma rqrstep-rename-vars: assumes R:  $\bigwedge$  st. st  $\in$  R  $\implies$   $\exists$  st'. st'  $\in$  R'  $\wedge$  st
 $=_v$  st'
  shows rqrstep nfs Q R  $\subseteq$  rqrstep nfs Q R'
proof
  fix x y
  assume (x,y)  $\in$  rqrstep nfs Q R
  from rqrstepE[OF this]
  obtain l r  $\sigma$  where NF:  $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q$  and x: x = l  $\cdot$   $\sigma$  and y: y
= r  $\cdot$   $\sigma$  and lr: (l,r)  $\in$  R
  and nfs: NF-subst nfs (l,r)  $\sigma$  Q .
  from R[OF lr] obtain l' r' where lr': (l',r')  $\in$  R' and eq: (l,r)  $=_v$  (l',r') by
auto
  from eq-rule-mod-varsE[OF eq]
  obtain  $\sigma 1$   $\sigma 2$  where 1: l = l'  $\cdot$   $\sigma 1$  r = r'  $\cdot$   $\sigma 1$  and 2: l' = l  $\cdot$   $\sigma 2$  r' = r  $\cdot$   $\sigma 2$ 
by auto
  from x y have xy: x = l'  $\cdot$  ( $\sigma 1 \circ_s \sigma$ ) y = r'  $\cdot$  ( $\sigma 1 \circ_s \sigma$ ) unfolding 1 by auto
  show (x,y)  $\in$  rqrstep nfs Q R'
  proof (rule rqrstepI[OF - lr' xy])
    show  $\forall u \triangleleft l' \cdot \sigma 1 \circ_s \sigma. u \in \text{NF-terms } Q$  using NF[unfolded 1] by simp
    show NF-subst nfs (l', r') ( $\sigma 1 \circ_s \sigma$ ) Q
  proof
    fix x
    assume nfs and x: x  $\in$  vars-term l'  $\vee$  x  $\in$  vars-term r'
    have l'  $\cdot$   $\sigma 1 \cdot \sigma 2$  = l' r'  $\cdot$   $\sigma 1 \cdot \sigma 2$  = r' unfolding 1[symmetric] 2[symmetric]
by auto
    then have l': l'  $\cdot$  ( $\sigma 1 \circ_s \sigma 2$ ) = l'  $\cdot$  Var and r': r'  $\cdot$  ( $\sigma 1 \circ_s \sigma 2$ ) = r'  $\cdot$  Var
by auto
    from term-subst-eq-rev[OF l'] term-subst-eq-rev[OF r'] x have ( $\sigma 1 \circ_s \sigma 2$ ) x

```

= *Var x* **by** *blast*
 then have *id*: $\sigma 1 \ x \cdot \sigma 2 = \text{Var } x$ **unfolding** *subst-compose-def* **by** *auto*
 then obtain *y* **where** $x1: \sigma 1 \ x = \text{Var } y$ **by** (*cases* $\sigma 1 \ x$, *auto*)
 from *x*
 have *y*: $y \in \text{vars-term } l \vee y \in \text{vars-term } r$ **unfolding** *1* **using** *x1* **unfolding**
vars-term-subst **by** *force*
 then have *y*: $y \in \text{vars-rule } (l, r)$ **unfolding** *vars-rule-def* **by** *simp*
 from *nfs*[*unfolded NF-subst-def*] $\langle \text{nfs} \rangle \ y$ **have** $\sigma \ y \in \text{NF-terms } Q$ **by** *auto*
 then show $(\sigma 1 \circ_s \sigma) \ x \in \text{NF-terms } Q$ **unfolding** *subst-compose-def* *x1* **by**
simp
 qed
 qed
 qed

lemma *qrstep-rename-vars*: **assumes** $R: \bigwedge st. st \in R \implies \exists st'. st' \in R' \wedge st =_v st'$
shows $\text{qrstep } \text{nfs } Q \ R \subseteq \text{qrstep } \text{nfs } Q \ R'$
proof
 fix *s t*
 assume $(s, t) \in \text{qrstep } \text{nfs } Q \ R$
 from *qrstepE*[*OF this*] **obtain** *C* $\sigma \ l \ r$ **where**
 $\forall u \triangleleft l \cdot \sigma. u \in \text{NF-terms } Q \ (l, r) \in R$ **and** $st: s = C \langle l \cdot \sigma \rangle \ t = C \langle r \cdot \sigma \rangle$ **and**
NF-subst *nfs* $(l, r) \ \sigma \ Q$.
 then have $(l \cdot \sigma, r \cdot \sigma) \in \text{rqrstep } \text{nfs } Q \ R$ **by** *auto*
 from *set-mp*[*OF rqrstep-rename-vars*[*OF R*] *this*] **have** $(l \cdot \sigma, r \cdot \sigma) \in \text{rqrstep}$
nfs $Q \ R'$.
 then have $(l \cdot \sigma, r \cdot \sigma) \in \text{qrstep } \text{nfs } Q \ R'$ **unfolding** *qrstep-iff-rqrstep-or-nrqrstep*
 ..
 with *st* **show** $(s, t) \in \text{qrstep } \text{nfs } Q \ R'$ **by** *auto*
qed

lemma *NF-subst-qrstep*: **assumes** $\bigwedge fn. fn \in \text{funas-term } t \implies \neg \text{defined } (\text{applicable-rules } Q \ R) \ fn$
and *varsNF*: $\bigwedge x. x \in \text{vars-term } t \implies \sigma \ x \in \text{NF } (\text{qrstep } \text{nfs } Q \ R)$
and *var-cond*: $\forall (l, r) \in R. \text{is-Fun } l$
shows $(t \cdot \sigma) \in \text{NF } (\text{qrstep } \text{nfs } Q \ R)$
proof
 fix *s*
 show $(t \cdot \sigma, s) \notin \text{qrstep } \text{nfs } Q \ R$
 proof
 assume $(t \cdot \sigma, s) \in \text{qrstep } \text{nfs } Q \ R$
 from *qrstepE*[*OF this*]
 obtain *C* $\sigma' \ l \ r$ **where** *nf*: $\forall u \triangleleft l \cdot \sigma'. u \in \text{NF-terms } Q$
 and *lr*: $(l, r) \in R$ **and** $t: t \cdot \sigma = C \langle l \cdot \sigma' \rangle$ **and** *nfs*: $\text{NF-subst } \text{nfs } (l, r) \ \sigma' \ Q$.
 from *var-cond* *lr*
 obtain *f ls* **where** $l: l = \text{Fun } f \ ls$ **by** (*cases* *l*, *auto*)
 let *?f* = (*f*, *length* *ls*)
 from *l lr only-applicable-rules*[*OF nf*] **have** *defined* (*applicable-rules* $Q \ R$) *?f*

```

    unfolding applicable-rules-def defined-def by force
  with assms have f: ?f  $\notin$  funas-term t by auto
  with varsNF t
  show False
  proof (induct t arbitrary: C)
    case (Var x)
    from Var(1)[of x] Var(2) have NF:  $C \langle l \cdot \sigma \rangle \in NF$  (qrstep nfs Q R) by
  auto
    with qrstepI[OF nf lr refl refl nfs, of C] show False by auto
  next
    case (Fun g ts)
    from Fun(3-4) arg-cong[OF Fun(3), of root, unfolded l, simplified] l obtain
  bef D aft where
    C:  $C = \text{More } g \text{ bef } D \text{ aft}$  and len:  $\text{length } ts = \text{Suc } (\text{length } bef + \text{length } aft)$ 
  by (cases C, auto)
    from Fun(3)[unfolded C] have id:  $\text{map } (\lambda t. t \cdot \sigma) \text{ } ts = bef @ D \langle l \cdot \sigma \rangle \# aft$ 
  by auto
    let ?i = length bef
    let ?t = ts ! ?i
    from len have mem: ?t  $\in$  set ts by auto
    from len arg-cong[OF id, of  $\lambda ts. ts ! ?i$ ] have idt: ?t  $\cdot \sigma = D \langle l \cdot \sigma' \rangle$  by
  simp
    show False
    by (rule Fun(1)[OF mem Fun(2) idt], insert Fun(4) mem, auto)
  qed
  qed
  qed
  lemma NF-subst-from-NF-args:
    assumes wf:  $Q \neq \{\}$   $\implies$  nfs  $\implies$  wf-rule (s,t)
    and NF:  $\text{set } (\text{args } (s \cdot \sigma)) \subseteq NF\text{-terms } Q$ 
    shows NF-subst nfs (s, t)  $\sigma$  Q
  proof (cases Q =  $\{\}$   $\vee$   $\neg$  nfs)
    case False
    with wf obtain f ss where s:  $s = \text{Fun } f \text{ } ss$  and vars:  $\text{vars-term } s \supseteq \text{vars-term } t$ 
    unfolding wf-rule-def by (cases s, auto)
    show ?thesis
  proof
    fix x
    assume x  $\in$  vars-term s  $\vee$  x  $\in$  vars-term t
    with vars have x  $\in$  vars-term s by auto
    with s obtain si where si: si  $\in$  set ss and x  $\in$  vars-term si by auto
    then have si  $\supseteq$  Var x by auto
    then have sub: si  $\cdot \sigma \supseteq \sigma$  x by auto
    from si NF s have si  $\cdot \sigma \in NF\text{-terms } Q$  by auto
    from NF-subterm[OF this sub] show  $\sigma$  x  $\in$  NF-terms Q .
  qed
  qed auto

```

end

5 The Critical Pair Lemma (with variables fixed to type string)

theory *Critical-Pairs*

imports

Q-Restricted-Rewriting

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* ·_c σ) \langle *r'* · τ \rangle , *l* · σ , *r* · σ) | *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 (σ, τ) }

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* ·_c σ) \langle *r'* · τ \rangle , *r* · σ) | *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 (σ, τ) }

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* (σ, τ) **and** *t* = *r* · σ
and *s* = (*C* ·_c σ) \langle *r'* · τ \rangle **and** *b* = (*C* = \square)
shows (*b*, *s*, *t*) $\in critical-pairs\ P\ R$
using *assms* **unfolding** *critical-pairs-def* **by** *blast*

lemma *critical-pairs-mono*:

assumes *S*₁ $\subseteq R₁ **and** *S*₂ $\subseteq R₂ **shows** *critical-pairs S*₁ *S*₂ $\subseteq critical-pairs\ R₁
*R*₂
unfolding *critical-pairs-def* **using** *assms* **by** *blast*$$$

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*) $\in (rstep\ R)^* \circ rstep\ P \circ ((rstep\ R)^*)^{\wedge -1} \vee$
($\exists\ C\ l\ m\ r\ \sigma. s = C\langle l \cdot \sigma \rangle \wedge t = C\langle m \cdot \sigma \rangle \wedge u = C\langle r \cdot \sigma \rangle \wedge$

```

    (l, m, r) ∈ critical-Peaks P R)
proof –
  let ?R = rstep R
  let ?CP = critical-Peaks P R
  from rrstepE[OF tu] obtain l1 r1 σ1 where lr1: (l1, r1) ∈ P and t1: t = l1
  · σ1 and u: u = r1 · σ1 by auto
  from ts obtain C l2 r2 σ2 where lr2: (l2, r2) ∈ R and t2: t = C⟨l2 · σ2⟩
and s: s = C⟨r2 · σ2⟩ by auto
  from t1 t2 have id: l1 · σ1 = C⟨l2 · σ2⟩ by auto
  let ?p = hole-pos C
  show ?thesis
proof (cases ?p ∈ poss l1 ∧ is-Fun (l1 |- ?p))
  case True
  then have p: ?p ∈ poss l1 by auto
  from ctxt-supt-id [OF p] obtain D where Dl1: D⟨l1 |- ?p⟩ = l1
  and D: D = ctxt-of-pos-term (hole-pos C) l1 by blast
  from arg-cong [OF Dl1, of λ t. t · σ1]
  have (D ·c σ1)⟨(l1 |- ?p) · σ1⟩ = C⟨l2 · σ2⟩ unfolding id by simp
  from arg-cong [OF this, of λ t. t |- ?p]
  have l2 · σ2 = (D ·c σ1)⟨(l1 |- ?p) · σ1⟩ |- ?p by simp
  also have ... = (D ·c σ1)⟨(l1 |- ?p) · σ1⟩ |- (hole-pos (D ·c σ1))
  using hole-pos-ctxt-of-pos-term [OF p] unfolding D by simp
  also have ... = (l1 |- ?p) · σ1 by (rule subt-at-hole-pos)
  finally have ident: l2 · σ2 = l1 |- ?p · σ1 by auto
  from mgu-vd-complete [OF ident [symmetric]]
  obtain μ1 μ2 ρ where mgu: mgu-vd ren (l1 |- ?p) l2 = Some (μ1, μ2) and
  μ1: σ1 = μ1 ∘s ρ
  and μ2: σ2 = μ2 ∘s ρ
  and ident': l1 |- ?p · μ1 = l2 · μ2 by blast
  have in-cp: ((D ·c μ1)⟨r2 · μ2⟩, l1 · μ1, r1 · μ1) ∈ ?CP
  unfolding critical-Peaks-def
  apply clarify
  apply (intro exI conjI)
  apply (rule refl)
  apply (rule lr1)
  apply (rule lr2)
  apply (rule Dl1[symmetric])
  apply (rule True[THEN conjunct2])
  apply (rule mgu)
  done
from hole-pos-ctxt-of-pos-term [OF p] D have pD: ?p = hole-pos D by simp
from id have C: C = ctxt-of-pos-term ?p (l1 · σ1) by simp
have C⟨r2 · σ2⟩ = (ctxt-of-pos-term ?p (l1 · σ1))⟨r2 · σ2⟩ using C by simp
also have ... = (ctxt-of-pos-term ?p l1 ·c σ1)⟨r2 · σ2⟩ unfolding ctxt-of-pos-term-subst
[OF p] ..
also have ... = (D ·c σ1)⟨r2 · σ2⟩ unfolding D ..
finally have Crσ: C⟨r2 · σ2⟩ = (D ·c σ1)⟨r2 · σ2⟩ .
show ?thesis unfolding Crσ s u t1 unfolding μ1 μ2
proof (rule disjI2, intro exI, intro conjI)

```

```

    show  $r1 \cdot \mu1 \circ_s \varrho = \Box \langle r1 \cdot \mu1 \cdot \varrho \rangle$  by simp
qed (insert in-cp, auto)
next
  case False
  from pos-into-subst [OF id - False]
  obtain  $q \ q' \ x$  where  $p: ?p = q @ q'$  and  $q: q \in \text{poss } l1$  and  $l1q: l1 \mid - q = \text{Var}$ 
x by auto
  have  $l2 \cdot \sigma2 = C \langle l2 \cdot \sigma2 \rangle \mid - (q @ q')$  unfolding p [symmetric] by simp
  also have  $\dots = l1 \cdot \sigma1 \mid - (q @ q')$  unfolding id ..
  also have  $\dots = l1 \mid - q \cdot \sigma1 \mid - q'$  using q by simp
  also have  $\dots = \sigma1 \ x \mid - q'$  unfolding l1q by simp
  finally have  $l2x: l2 \cdot \sigma2 = \sigma1 \ x \mid - q'$  by simp
  have  $pp: ?p \in \text{poss } (l1 \cdot \sigma1)$  unfolding id by simp
  then have  $q @ q' \in \text{poss } (l1 \cdot \sigma1)$  unfolding p .
  then have  $q' \in \text{poss } (l1 \cdot \sigma1 \mid - q)$  unfolding poss-append-poss ..
  with q have  $q' \in \text{poss } (l1 \mid - q \cdot \sigma1)$  by auto
  then have  $q'x: q' \in \text{poss } (\sigma1 \ x)$  unfolding l1q by simp
  from ctxt-supt-id [OF q'x] obtain  $E$  where  $\sigma1x: E \langle l2 \cdot \sigma2 \rangle = \sigma1 \ x$ 
    and  $E: E = \text{ctxt-of-pos-term } q' (\sigma1 \ x)$ 
    unfolding l2x by blast
  let  $?e = E \langle r2 \cdot \sigma2 \rangle$ 
  from hole-pos-ctxt-of-pos-term [OF q'x] E have  $q': q' = \text{hole-pos } E$  by simp
  from  $\sigma1x$  have  $\sigma1x': \sigma1 \ x = E \langle l2 \cdot \sigma2 \rangle$  by simp
  let  $?s = \lambda y. \text{if } y = x \text{ then } ?e \text{ else } \sigma1 \ y$ 
  have  $(u, r1 \cdot ?s) \in (\text{rstep } R)^*$  unfolding u
proof (rule subst-qsteps-imp-qsteps [where  $Q = \{\}$ , unfolded qstep-rstep-conv])
  fix y
  show  $(\sigma1 \ y, ?s \ y) \in (\text{rstep } R)^*$ 
  proof (cases  $y = x$ )
    case True
    show ?thesis unfolding True  $\sigma1x'$  using lr2 by auto
  qed simp
qed
hence  $r1u: (r1 \cdot ?s, u) \in ((\text{rstep } R)^*)^{\wedge -1}$  by auto
show ?thesis
proof (rule disjI1, intro relcompI)
  show  $(r1 \cdot ?s, u) \in ((\text{rstep } R)^*)^{\wedge -1}$  by fact
  show  $(l1 \cdot ?s, r1 \cdot ?s) \in \text{rrstep } P$  using lr1 by auto
  from q have  $ql1: q \in \text{poss } (l1 \cdot \sigma1)$  by simp
  have  $s = \text{replace-at } (C \langle l2 \cdot \sigma2 \rangle) \ ?p \ (r2 \cdot \sigma2)$  unfolding s by simp
  also have  $\dots = \text{replace-at } (l1 \cdot \sigma1) \ ?p \ (r2 \cdot \sigma2)$  unfolding id ..
  also have  $\dots = \text{replace-at } (l1 \cdot \sigma1) \ q \ ?e$ 
  proof -
    have  $E = \text{ctxt-of-pos-term } q' (l1 \cdot \sigma1 \mid - q)$ 
    unfolding subt-at-subst [OF q] l1q E by simp
    then show ?thesis
    unfolding p
    unfolding ctxt-of-pos-term-append [OF ql1]
    by simp
  qed

```

```

qed
finally have s: s = replace-at (l1 · σ1) q ?e .
from q l1q have (replace-at (l1 · σ1) q ?e, l1 · ?σ) ∈ ?R∧*
proof (induct l1 arbitrary: q)
  case (Fun f ls)
    from Fun(2, 3) obtain i p where q: q = i # p and i: i < length ls and
p: p ∈ poss (ls ! i) and px: ls ! i |- p = Var x by (cases q, auto)
    from i have ls ! i ∈ set ls by auto
    from Fun(1) [OF this p px] have rec: (replace-at (ls ! i · σ1) p ?e, ls ! i ·
?σ) ∈ ?R∧* .
    let ?lsσ = map (λ t. t · σ1) ls
    let ?lsσ' = map (λ t. t · ?σ) ls
    have id: replace-at (Fun f ls · σ1) q ?e = Fun f (take i ?lsσ @ replace-at
(ls ! i · σ1) p ?e # drop (Suc i) ?lsσ) (is - = Fun f ?r)
    unfolding q using i by simp
    show ?case unfolding id unfolding eval-term.simps
    proof (rule all-ctxt-closedD [of UNIV])
      fix j
      assume j: j < length ?r
      show (?r ! j, ?lsσ' ! j) ∈ ?R∧*
      proof (cases j = i)
        case True
          show ?thesis using i True using rec by (auto simp: nth-append)
        case False
          have ?r ! j = ?lsσ ! j
            by (rule nth-append-take-drop-is-nth-conv, insert False i j, auto)
          also have ... = ls ! j · σ1 using j i by auto
          finally have idr: ?r ! j = ls ! j · σ1 .
          from j i have idl: ?lsσ' ! j = ls ! j · ?σ by auto
          show ?thesis unfolding idr idl
        proof (rule subst-qsteps-imp-qsteps [where Q = {}, unfolded qstep-rstep-conv])
          fix y
          show (σ1 y, ?σ y) ∈ ?R∧*
          proof (cases y = x)
            case True then show ?thesis using σ1x' lr2 by auto
          qed simp
        qed
      qed
    qed
  qed (insert i, auto)
qed simp
then show (s, l1 · ?σ) ∈ ?R∧* unfolding s .
qed
qed
qed

```

lemma critical-Peaks-main-rrstep:

fixes R :: ('f, 'v) trs

assumes tu: (t, u) ∈ rstep R and ts: (t, s) ∈ rstep R

shows $(s, u) \in \text{join } (\text{rstep } R) \vee$
 $(\exists C \, l \, m \, r \, \sigma. s = C \langle l \cdot \sigma \rangle \wedge t = C \langle m \cdot \sigma \rangle \wedge u = C \langle r \cdot \sigma \rangle \wedge$
 $(l, m, r) \in \text{critical-Peaks } R \, R)$
using *critical-Peaks-main*[*OF assms*]
proof
assume $(s, u) \in (\text{rstep } R)^* \, O \, \text{rrstep } R \, O \, ((\text{rstep } R)^*)^{-1}$
also have $\dots \subseteq (\text{rstep } R)^* \, O \, ((\text{rstep } R)^*)^{-1}$
unfolding *rstep-iff-rrstep-or-nrrstep* **by** *regexp*
finally have $(s, u) \in \text{join } (\text{rstep } R)$ **by** *blast*
thus *?thesis* **by** *auto*
qed *auto*

lemma *critical-Peaks-main-rstep*:
fixes $R :: ('f, 'v) \, \text{trs}$
assumes $tu: (t, u) \in \text{rstep } R$ **and** $ts: (t, s) \in \text{rstep } R$
shows $(s, u) \in \text{join } (\text{rstep } R) \vee$
 $(\exists C \, l \, m \, r \, \sigma. s = C \langle l \cdot \sigma \rangle \wedge t = C \langle m \cdot \sigma \rangle \wedge u = C \langle r \cdot \sigma \rangle \wedge$
 $((l, m, r) \in \text{critical-Peaks } R \, R \vee (r, m, l) \in \text{critical-Peaks } R \, R))$
proof –
let $?R = \text{rstep } R$
let $?CP = \text{critical-Peaks } R \, R$
from tu **obtain** $C1 \, l1 \, r1 \, \sigma1$ **where** $lr1: (l1, r1) \in R$ **and** $t1: t = C1 \langle l1 \cdot \sigma1 \rangle$
and $u: u = C1 \langle r1 \cdot \sigma1 \rangle$ **by** *auto*
from ts **obtain** $C2 \, l2 \, r2 \, \sigma2$ **where** $lr2: (l2, r2) \in R$ **and** $t2: t = C2 \langle l2 \cdot \sigma2 \rangle$
and $s: s = C2 \langle r2 \cdot \sigma2 \rangle$ **by** *auto*
define n **where** $n = \text{size } C1 + \text{size } C2$
from $t1 \, t2 \, u \, s$ $n\text{-def}$ **show** *?thesis*
proof (*induct n arbitrary: C1 C2 s t u rule: less-induct*)
case (*less n C1 C2 s t u*)
show *?case*
proof (*cases C1*)
case *Hole*
with *less(2,4) lr1* **have** $tu: (t, u) \in \text{rrstep } R$ **by** *auto*
from *less(3,5) lr2* **have** $ts: (t, s) \in \text{rstep } R$ **by** *auto*
from *critical-Peaks-main-rrstep*[*OF tu ts*] **show** *?thesis* **by** *auto*
next
case (*More f1 bef1 D1 aft1*) **note** $C1 = \text{this}$
show *?thesis*
proof (*cases C2*)
case *Hole*
with *less(3,5) lr2* **have** $ts: (t, s) \in \text{rrstep } R$ **by** *auto*
from *less(2,4) lr1* **have** $tu: (t, u) \in \text{rstep } R$ **by** *auto*
from *critical-Peaks-main-rrstep*[*OF ts tu*] **show** *?thesis* **by** *auto*
next
case (*More f2 bef2 D2 aft2*) **note** $C2 = \text{this}$
from *less(2-3) C1 C2*
have $\text{id}: (\text{More } f1 \, \text{bef1 } D1 \, \text{aft1}) \langle l1 \cdot \sigma1 \rangle = (\text{More } f2 \, \text{bef2 } D2 \, \text{aft2}) \langle l2 \cdot \sigma2 \rangle$
by *auto*
let $?n1 = \text{length } \text{bef1}$

```

let ?n2 = length bef2
from id have f: f1 = f2 by simp
show ?thesis
proof (cases ?n1 = ?n2)
  case True
  with id have idb: bef1 = bef2 and ida: aft1 = aft2
    and idD: D1⟨l1 · σ1⟩ = D2⟨l2 · σ2⟩ by auto
  let ?C = More f2 bef2 □ aft2
  have id1: C1 = ?C ◦c D1 unfolding C1 f ida idb by simp
  have id2: C2 = ?C ◦c D2 unfolding C2 by simp
  define m where m = size D1 + size D2
  have mn: m < n unfolding less m-def C1 C2 by auto
  note IH = less(1)[OF mn refl idD refl refl m-def]
  then show ?thesis
  proof
    assume ( D2⟨r2 · σ2⟩, D1⟨r1 · σ1⟩) ∈ join ?R
    then obtain s' where seq1: (D1⟨r1 · σ1⟩, s') ∈ ?R∧*
      and seq2: (D2⟨r2 · σ2⟩, s') ∈ ?R∧* by auto
    from rsteps-closed-ctxt [OF seq1, of ?C]
    have seq1: (C1⟨r1 · σ1⟩, ?C⟨s'⟩) ∈ ?R∧* using id1 by auto
    from rsteps-closed-ctxt [OF seq2, of ?C]
    have seq2: (C2⟨r2 · σ2⟩, ?C⟨s'⟩) ∈ ?R∧* using id2 by auto
    from seq1 seq2 show ?thesis using less by auto
  next
    assume ∃ C l m r σ. D2⟨r2 · σ2⟩ = C⟨l · σ⟩ ∧ D1⟨l1 · σ1⟩ = C⟨m · σ⟩
    ∧ D1⟨r1 · σ1⟩ = C⟨r · σ⟩ ∧ ((l, m, r) ∈ ?CP ∨ (r, m, l) ∈ ?CP)
    then obtain C l m r σ where idD: D2⟨r2 · σ2⟩ = C⟨l · σ⟩ D1⟨l1 · σ1⟩ = C⟨m · σ⟩
    D1⟨r1 · σ1⟩ = C⟨r · σ⟩ and mem: ((l, m, r) ∈ ?CP ∨ (r, m, l) ∈ ?CP) by blast
    show ?thesis
      apply (intro disjI2)
      apply (unfold less id1 id2)
      apply (intro exI [of - ?C ◦c C] exI)
      by (rule conjI [OF - conjI [OF - conjI[OF - mem]]], insert idD, auto)
  qed
next
case False
let ?p1 = ?n1 # hole-pos D1
let ?p2 = ?n2 # hole-pos D2
have l2: C1⟨l1 · σ1⟩ |- ?p2 = l2 · σ2 unfolding C1 id by simp
have p12: ?p1 ⊥ ?p2 using False by simp
have p1: ?p1 ∈ poss (C1⟨l1 · σ1⟩) unfolding C1 by simp
have p2: ?p2 ∈ poss (C1⟨l1 · σ1⟩) unfolding C1 unfolding id by simp
let ?one = replace-at (C1⟨l1 · σ1⟩) ?p1 (r1 · σ1)
have one: C1⟨r1 · σ1⟩ = ?one unfolding C1 by simp
from parallel-qstep [OF p12 p1 p2 l2 - lr2, of {} True r1 · σ1]
have (?one, replace-at ?one ?p2 (r2 · σ2)) ∈ rstep R by auto
then have one: (C1⟨r1 · σ1⟩, replace-at ?one ?p2 (r2 · σ2)) ∈ (rstep
R)∧* unfolding one by simp

```

have $l1: C2\langle l2 \cdot \sigma2 \rangle \mid - ?p1 = l1 \cdot \sigma1$ **unfolding** $C2\ id$ [*symmetric*] **by**
simp
have $p21: ?p2 \perp ?p1$ **using** *False* **by** *simp*
have $p1': ?p1 \in \text{poss}(C2\langle l2 \cdot \sigma2 \rangle)$ **unfolding** $C2\ id$ [*symmetric*] **by** *simp*
have $p2': ?p2 \in \text{poss}(C2\langle l2 \cdot \sigma2 \rangle)$ **unfolding** $C2$ **by** *simp*
let $?two = \text{replace-at}(C2\langle l2 \cdot \sigma2 \rangle)\ ?p2\ (r2 \cdot \sigma2)$
have $two: C2\langle r2 \cdot \sigma2 \rangle = ?two$ **unfolding** $C2$ **by** *simp*
from *parallel-qstep* [*OF* $p21\ p2'\ p1'\ l1 - lr1$, *of* $\{\}$ *True* $r2 \cdot \sigma2$]
have $(?two, \text{replace-at}\ ?two\ ?p1\ (r1 \cdot \sigma1)) \in \text{rstep}\ R$ **by** *auto*
then have $two: (C2\langle r2 \cdot \sigma2 \rangle, \text{replace-at}\ ?two\ ?p1\ (r1 \cdot \sigma1)) \in (\text{rstep}\ R)^*$ **unfolding** two **by** *simp*
have $\text{replace-at}\ ?one\ ?p2\ (r2 \cdot \sigma2) = \text{replace-at}\ (\text{replace-at}\ (C1\langle l1 \cdot \sigma1 \rangle)\ ?p2\ (r2 \cdot \sigma2))\ ?p1\ (r1 \cdot \sigma1)$
by (*rule parallel-replace-at* [*OF* $p12\ p1\ p2$])
also have $... = \text{replace-at}\ ?two\ ?p1\ (r1 \cdot \sigma1)$ **unfolding** $C1\ C2\ id\ ..$
finally have $\text{one-two: replace-at}\ ?one\ ?p2\ (r2 \cdot \sigma2) = \text{replace-at}\ ?two\ ?p1\ (r1 \cdot \sigma1)$.
show *?thesis* **unfolding** *less*
by (*rule disjI1*, *insert one one-two two*, *auto*)
qed
qed
qed
qed
qed

lemma *critical-Peak-steps*:
fixes $R :: ('f, 'v)\ trs$ **and** S
assumes $cp: (l, m, r) \in \text{critical-Peaks}\ R\ S$
shows $(m, l) \in \text{rstep}\ S\ (m, r) \in \text{rstep}\ R\ (m, r) \in \text{rrstep}\ R$
proof –
from cp [*unfolded critical-Peaks-def*]
obtain $\sigma1\ \sigma2\ l1\ l2\ r1\ r2\ C$ **where** $id: r = r1 \cdot \sigma1\ l = (C \cdot_c \sigma1)\langle r2 \cdot \sigma2 \rangle\ m = (C \cdot_c \sigma1)\langle l1 \cdot \sigma1 \rangle$
and $r1: (C\langle l1 \rangle, r1) \in R$ **and** $r2: (l2, r2) \in S$ **and** $mgu: mgu\text{-vd}\ \text{ren}\ l1\ l2 = \text{Some}\ (\sigma1, \sigma2)$ **by** *auto*
have $(C\langle l1 \rangle \cdot \sigma1, r) \in \text{rrstep}\ R$ **unfolding** id
by (*rule rrstepI* [*of* $C\langle l1 \rangle\ r1 - - \sigma1$] $r1$, *insert* $r1$, *auto*)
thus $(m, r) \in \text{rrstep}\ R$ **unfolding** id **by** *auto*
thus $(m, r) \in \text{rstep}\ R$ **by** (*rule rrstep-imp-rstep*)
from $mgu\text{-vd-sound}$ [*OF* mgu] **have** $\text{change: } C\langle l1 \rangle \cdot \sigma1 = (C \cdot_c \sigma1)\langle l2 \cdot \sigma2 \rangle$
by *simp*
have $(C\langle l1 \rangle \cdot \sigma1, l) \in \text{rstep}\ S$ **unfolding** $\text{change}\ id$
by (*rule rstepI* [*OF* $r2$, *of* $- - \sigma2$], *auto*)
thus $(m, l) \in \text{rstep}\ S$ **unfolding** id **by** *auto*
qed

lemma *critical-Peak-to-pair*: **assumes** $(l, m, r) \in \text{critical-Peaks}\ R\ R$
shows $\exists\ b. (b, l, r) \in \text{critical-pairs}\ R\ R$
using *assms* **unfolding** *critical-Peaks-def* *critical-pairs-def* **by** *blast*

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 $(t1, t2) \in \text{join } (\text{rstep } R) \vee$
 $(\exists C \ b \ l \ r \ \sigma. t1 = C\langle l \cdot \sigma \rangle \wedge t2 = C\langle r \cdot \sigma \rangle \wedge$
 $((b, l, r) \in \text{critical-pairs } R \ R \vee (b, r, l) \in \text{critical-pairs } R \ R))$

using *critical-Peaks-main-rstep*[*OF st2 st1*]

proof

assume $\exists C \ l \ m \ r \ \sigma.$
 $t1 = C\langle l \cdot \sigma \rangle \wedge s = C\langle m \cdot \sigma \rangle \wedge t2 = C\langle r \cdot \sigma \rangle \wedge ((l, m, r) \in \text{critical-Peaks}$
 $R \ R \vee (r, m, l) \in \text{critical-Peaks } R \ R)$

then obtain $C \ l \ m \ r \ \sigma$ **where** $id: t1 = C\langle l \cdot \sigma \rangle \ t2 = C\langle r \cdot \sigma \rangle$ **and** $disj: ((l,$
 $m, r) \in \text{critical-Peaks } R \ R \vee (r, m, l) \in \text{critical-Peaks } R \ R)$

by *blast*

from *critical-Peak-to-pair disj* **obtain** b **where** $(b, l, r) \in \text{critical-pairs } R \ R \vee$
 $(b, r, l) \in \text{critical-pairs } R \ R$ **by** *blast*

with id **show** *?thesis* **by** *blast*

qed *auto*

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 $WCR \ (\text{rstep } R)$

proof

let $?R = \text{rstep } R$

let $?CP = \text{critical-pairs } R \ R$

fix $s \ t1 \ t2$

assume $steps: (s, t1) \in ?R \ (s, t2) \in ?R$

let $?p = \lambda s'. (t1, s') \in ?R^* \wedge (t2, s') \in ?R^*$

from *critical-pairs-main* [*OF steps*]

have $\exists s'. ?p \ s'$

proof

assume $\exists C \ b \ l \ r \ \sigma. t1 = C\langle l \cdot \sigma \rangle \wedge t2 = C\langle r \cdot \sigma \rangle \wedge ((b, l, r) \in ?CP \vee (b,$
 $r, l) \in ?CP)$

then obtain $C \ b \ l \ r \ \sigma$ **where** $id: t1 = C\langle l \cdot \sigma \rangle \ t2 = C\langle r \cdot \sigma \rangle$
and $mem: (b, l, r) \in ?CP \vee (b, r, l) \in ?CP$ **by** *blast*

show *?thesis*

proof (*cases l = r*)

case *True*

then show *?thesis* **unfolding** id **by** *auto*

next

case *False*

note $sub-ctxt = \text{rsteps-closed-ctxt} \ [\text{OF } \text{rsteps-closed-subst} \ [\text{OF } \text{rsteps-closed-subst}]]$

from mem **show** *?thesis*

proof

assume $mem: (b, l, r) \in ?CP$

from cp [OF mem $False$] obtain $l' r' s' \tau$ where $id2: l = l' \cdot \tau \ r = r' \cdot \tau$
 and $steps: (l', s') \in ?R^* \ (r', s') \in ?R^*$
 unfolding $instance-rule-def$ by $auto$
 show $\exists s'. ?p \ s'$ unfolding $id \ id2$
 by ($rule \ exI$ [$of - C(s' \cdot \tau \cdot \sigma)$], $rule \ conjI$, $rule \ sub-ctxt$ [$OF \ steps(1)$],
 $rule \ sub-ctxt$ [$OF \ steps(2)$])
 next
 assume $mem: (b, r, l) \in ?CP$
 from cp [OF mem] $False$ obtain $l' r' s' \tau$ where $id2: r = l' \cdot \tau \ l = r' \cdot \tau$
 and $steps: (l', s') \in ?R^* \ (r', s') \in ?R^*$
 unfolding $instance-rule-def$ by $auto$
 show $\exists s'. ?p \ s'$ unfolding $id \ id2$
 by ($rule \ exI$ [$of - C(s' \cdot \tau \cdot \sigma)$], $rule \ conjI$, $rule \ sub-ctxt$ [$OF \ steps(2)$],
 $rule \ sub-ctxt$ [$OF \ steps(1)$])
 qed
 qed
 then show $(t1, t2) \in join \ ?R$ by $auto$
 qed

lemma *critical-pairs-fork*:
 fixes $R :: ('f, 'v) \ trs$ and S
 assumes $cp: (b, l, r) \in critical-pairs \ R \ S$
 shows $(r, l) \in (rstep \ R)^{-1} \ O \ rstep \ S$
proof –
 from cp obtain m where $(l, m, r) \in critical-Peaks \ R \ S$
 unfolding $critical-pairs-def \ critical-Peaks-def$ by $blast$
 from $critical-Peak-steps(1-2)$ [$OF \ this$] show $?thesis$ by $auto$
 qed

lemma *critical-pairs-fork'*: assumes $(b, l, r) \in critical-pairs \ R \ S$
 shows $(l, r) \in (rstep \ S)^{-1} \ O \ rstep \ R$
 using *critical-pairs-fork* [$OF \ assms$] by $auto$

lemma *critical-pairs-complete*:
 fixes $R :: ('f, 'v) \ trs$
 assumes $cp: (b, l, r) \in critical-pairs \ R \ R$
 and $no-join: (l, r) \notin (rstep \ R)^\downarrow$
 shows $\neg \ WCR \ (rstep \ R)$
proof
 from *critical-pairs-fork* [$OF \ cp$] obtain u where $ul: (u, l) \in rstep \ R$ and $ur:$
 $(u, r) \in rstep \ R$ by $force$
 assume $wcr: WCR \ (rstep \ R)$
 with $ul \ ur \ no-join$ show $False$ unfolding $WCR-on-def$ by $auto$
 qed

lemma *critical-pair-lemma*:
 fixes $R :: ('f, 'v) \ trs$

shows $WCR (rstep R) \longleftrightarrow$
 $(\forall (b, s, t) \in critical-pairs R R. (s, t) \in (rstep R)^\downarrow)$
(is ?l = ?r)
proof
assume ?l
with *critical-pairs-complete* [where $R = R$] **show** ?r **by** *auto*
next
assume ?r
show ?l
proof (*rule critical-pairs*)
fix b s t
assume $(b, s, t) \in critical-pairs R R$
with $\langle ?r \rangle$ **have** $(s, t) \in join (rstep R)$ **by** *auto*
then obtain u **where** $s: (s, u) \in (rstep R)^{\wedge*}$
and $t: (t, u) \in (rstep R)^{\wedge*}$ **by** *auto*
show $\exists s' t' u. instance-rule (s, t) (s', t') \wedge (s', u) \in (rstep R)^{\wedge*} \wedge (t', u) \in (rstep R)^{\wedge*}$
proof (*intro exI conjI*)
show *instance-rule* (s, t) (s, t) **by** *simp*
qed (*insert s t, auto*)
qed
qed

lemma *critical-pairs-innermost-weak-diamond:*

fixes $R :: ('f, 'v) trs$
assumes $cp: \bigwedge l r. (True, l, r) \in critical-pairs R R \implies l = r$
and $NF-Q-R: NF-terms Q \subseteq NF-trs R$
and $lhss: \bigwedge l r. (l, r) \in R \implies is-Fun l$
shows $w\Diamond (qrstep nfs Q R)$
unfolding *weak-diamond-def*
proof
fix t1 t2 :: ('f, 'v) term
let ?R = *qrstep nfs Q R*
assume $(t1, t2) \in ?R^{\sim-1} O ?R - Id$
then obtain s **where** $st1: (s, t1) \in ?R$ **and** $st2: (s, t2) \in ?R$ **and** $t1 \neq t2$ **by** *auto*
let ?Q = *NF-terms Q*
from st1 **obtain** C1 l1 r1 $\sigma 1$ **where** $lr1: (l1, r1) \in R$ **and** $s1: s = C1 \langle l1 \cdot \sigma 1 \rangle$
and $t1: t1 = C1 \langle r1 \cdot \sigma 1 \rangle$
and $NF1: \forall u \triangleleft l1 \cdot \sigma 1. u \in ?Q$ **and** $nfs1: NF-subst nfs (l1, r1) \sigma 1 Q$ **by** *auto*
from st2 **obtain** C2 l2 r2 $\sigma 2$ **where** $lr2: (l2, r2) \in R$ **and** $s2: s = C2 \langle l2 \cdot \sigma 2 \rangle$
and $t2: t2 = C2 \langle r2 \cdot \sigma 2 \rangle$
and $NF2: \forall u \triangleleft l2 \cdot \sigma 2. u \in ?Q$ **and** $nfs2: NF-subst nfs (l2, r2) \sigma 2 Q$ **by** *auto*
from s1 s2 **have** $id: C1 \langle l1 \cdot \sigma 1 \rangle = C2 \langle l2 \cdot \sigma 2 \rangle$ **by** *simp*
let ?p = $\lambda (C1, C2). C1 \langle l1 \cdot \sigma 1 \rangle = C2 \langle l2 \cdot \sigma 2 \rangle \longrightarrow (\exists s'. (C1 \langle r1 \cdot \sigma 1 \rangle, s') \in ?R \wedge (C2 \langle r2 \cdot \sigma 2 \rangle, s') \in ?R \vee C1 \langle r1 \cdot \sigma 1 \rangle = C2 \langle r2 \cdot \sigma 2 \rangle)$

```

{
  fix C12
  let ?m =  $\lambda$  (C1, C2). size C1 + size C2
  have ?p C12
  proof (induct rule: wf-induct [OF wf-measure [of ?m], of ?p])
    case (1 C12)
    obtain C1 C2 where C12: C12 = (C1, C2) by force
    show ?p C12 unfolding C12 split
    proof (intro impI)
      assume id: C1⟨l1 ·  $\sigma$ 1⟩ = C2⟨l2 ·  $\sigma$ 2⟩
      show  $\exists$  s'. ( (C1⟨r1 ·  $\sigma$ 1⟩, s')  $\in$  ?R  $\wedge$  (C2⟨r2 ·  $\sigma$ 2⟩, s')  $\in$  ?R  $\vee$  C1⟨r1 ·
 $\sigma$ 1⟩ = C2⟨r2 ·  $\sigma$ 2⟩)
      proof (cases C1)
        case Hole note C1 = this
        with id have id: l1 ·  $\sigma$ 1 = C2⟨l2 ·  $\sigma$ 2⟩ by simp
        have C2: C2 =  $\square$ 
        proof (rule ccontr)
          assume C2  $\neq$   $\square$ 
          with id have l2 ·  $\sigma$ 2  $\triangleleft$  l1 ·  $\sigma$ 1 by auto
          with NF1 NF-Q-R have l2 ·  $\sigma$ 2  $\in$  NF-trs R by auto
          with lr2 show False by auto
        qed
        with id have ident: l1 ·  $\sigma$ 1 = l2 ·  $\sigma$ 2 by simp
        from lhss [OF lr1] have nvar: is-Fun l1 .
        from mgu- $\nu$ d-complete [OF ident]
        obtain  $\mu$ 1  $\mu$ 2  $\varrho$  where mgu: mgu- $\nu$ d ren l1 l2 = Some ( $\mu$ 1,  $\mu$ 2) and
           $\mu$ 1:  $\sigma$ 1 =  $\mu$ 1  $\circ_s$   $\varrho$ 
          and  $\mu$ 2:  $\sigma$ 2 =  $\mu$ 2  $\circ_s$   $\varrho$ 
          by blast
        have in-cp: (True, r2 ·  $\mu$ 2, r1 ·  $\mu$ 1)  $\in$  critical-pairs R R
          by (rule critical-pairsI [OF lr1 lr2 - nvar mgu, of  $\square$ ], auto)
        from C2 have C2r $\sigma$ : C2⟨r2 ·  $\sigma$ 2⟩ = r2 ·  $\sigma$ 2 by simp
        from C1 have C1r $\sigma$ : C1⟨r1 ·  $\sigma$ 1⟩ = r1 ·  $\sigma$ 1 by simp
        from cp [OF in-cp, unfolded instance-rule-def] have id: r1 ·  $\mu$ 1 = r2 ·
 $\mu$ 2 ..
        from C1r $\sigma$  have C1⟨r1 ·  $\sigma$ 1⟩ = r2 ·  $\sigma$ 2 unfolding  $\mu$ 1 using id  $\mu$ 2 by
simp
        also have ... = C2⟨r2 ·  $\sigma$ 2⟩ unfolding C2r $\sigma$  ..
        finally show ?thesis by simp
      next
        case (More f1 bef1 D1 aft1) note C1 = this
        show ?thesis
        proof (cases C2)
          case Hole
          with id have l2 ·  $\sigma$ 2 = C1⟨l1 ·  $\sigma$ 1⟩ by auto
          with C1 have l1 ·  $\sigma$ 1  $\triangleleft$  l2 ·  $\sigma$ 2 by auto
          with NF2 NF-Q-R have l1 ·  $\sigma$ 1  $\in$  NF-trs R by auto
          with lr1 have False by auto
          then show ?thesis ..
        qed
      qed
    qed
  qed
}

```

```

next
  case (More f2 bef2 D2 aft2) note C2 = this
  let ?n1 = length bef1
  let ?n2 = length bef2
  note id = id [unfolded C1 C2]
  from id have f: f1 = f2 by simp
  show ?thesis
  proof (cases ?n1 = ?n2)
    case True
    with id have idb: bef1 = bef2 and ida: aft1 = aft2
    and idD: D1⟨l1 · σ1⟩ = D2⟨l2 · σ2⟩ by auto
    have ((D1, D2), C12) ∈ measure ?m unfolding C12 C1 C2
    by auto
    from 1 [rule-format, OF this, unfolded split, rule-format,
    OF idD] obtain s'
    where disj: (D1⟨r1 · σ1⟩, s') ∈ ?R ∧ (D2⟨r2 · σ2⟩, s') ∈ ?R ∨
    D1⟨r1 · σ1⟩ = D2⟨r2 · σ2⟩ (is ?seq1 ∧ ?seq2 ∨ ?id) by auto
    let ?C = More f2 bef2 □ aft2
    have id1: C1 = ?C ◦c D1 unfolding C1 f ida idb by simp
    have id2: C2 = ?C ◦c D2 unfolding C2 by simp
    from disj show ?thesis
    proof
      assume ?seq1 ∧ ?seq2
      then have seq1: ?seq1 and seq2: ?seq2 by auto
      from qrstep.ctx [OF seq1, of ?C]
      have seq1: (C1⟨r1 · σ1⟩, ?C⟨s'⟩) ∈ ?R using id1 by auto
      from qrstep.ctx [OF seq2, of ?C]
      have seq2: (C2⟨r2 · σ2⟩, ?C⟨s'⟩) ∈ ?R using id2 by auto
      from seq1 seq2 show ?thesis by auto
    next
      assume ?id
      then show ?thesis unfolding id1 id2 by simp
    qed
  next
  case False
  let ?p1 = ?n1 # hole-pos D1
  let ?p2 = ?n2 # hole-pos D2
  have l2: C1⟨l1 · σ1⟩ |- ?p2 = l2 · σ2 unfolding C1 id by simp
  have p12: ?p1 ⊥ ?p2 using False by simp
  have p1: ?p1 ∈ poss (C1⟨l1 · σ1⟩) unfolding C1 by simp
  have p2: ?p2 ∈ poss (C1⟨l1 · σ1⟩) unfolding C1 unfolding id by
simp
  let ?one = replace-at (C1⟨l1 · σ1⟩) ?p1 (r1 · σ1)
  have one: C1⟨r1 · σ1⟩ = ?one unfolding C1 by simp
  from parallel-qrstep [OF p12 p1 p2 l2 NF2 lr2 nfs2]
  have (?one, replace-at ?one ?p2 (r2 · σ2)) ∈ qrstep nfs Q R .
  then have one: (C1⟨r1 · σ1⟩, replace-at ?one ?p2 (r2 · σ2)) ∈ qrstep
nfs Q R unfolding one by simp
  have l1: C2⟨l2 · σ2⟩ |- ?p1 = l1 · σ1 unfolding C2 id [symmetric]

```

```

by simp
  have p21: ?p2 ⊥ ?p1 using False by simp
  have p1': ?p1 ∈ poss (C2⟨l2 · σ2⟩) unfolding C2 id [symmetric] by
simp
  have p2': ?p2 ∈ poss (C2⟨l2 · σ2⟩) unfolding C2 by simp
  let ?two = replace-at (C2⟨l2 · σ2⟩) ?p2 (r2 · σ2)
  have two: C2⟨r2 · σ2⟩ = ?two unfolding C2 by simp
  from parallel-qstep [OF p21 p2' p1' l1 NF1 lr1 nfs1]
  have (?two, replace-at ?two ?p1 (r1 · σ1)) ∈ qstep nfs Q R .
  then have two: (C2⟨r2 · σ2⟩, replace-at ?two ?p1 (r1 · σ1)) ∈ qstep
nfs Q R unfolding two by simp
  have replace-at ?one ?p2 (r2 · σ2) = replace-at (replace-at (C1⟨l1 ·
σ1⟩) ?p2 (r2 · σ2)) ?p1 (r1 · σ1)
  by (rule parallel-replace-at [OF p12 p1 p2])
  also have ... = replace-at ?two ?p1 (r1 · σ1) unfolding C1 C2 id ..
  finally have one-two: replace-at ?one ?p2 (r2 · σ2) = replace-at ?two
?p1 (r1 · σ1) .
  show ?thesis
  by (intro exI disjI1 conjI, rule one, unfold one-two, rule two)
qed
qed
qed
qed
qed
qed
}
from this [of (C1, C2), unfolded split, rule-format, OF id]
show (t1, t2) ∈ ?R O ?R-1 using t12 unfolding t1 t2 by auto
qed

lemma critical-pairs-innermost:
  assumes ∧ l r. (True, l, r) ∈ critical-pairs R R ⇒ l = r
  and NF-terms Q ⊆ NF-trs R
  and ∧ l r. (l, r) ∈ R ⇒ is-Fun l
  shows CR (qstep nfs Q R)
  by (rule weak-diamond-imp-CR [OF critical-pairs-innermost-weak-diamond [OF
assms]])

lemma critical-pairs-exI:
  fixes σ :: ('f, 'v) subst
  assumes P: (l, r) ∈ P and R: (l', r') ∈ R and l: l = C⟨l'⟩
  and l'': is-Fun l'' and unif: l'' · σ = l' · τ
  and b: b = (C = □)
  shows ∃ s t. (b, s, t) ∈ critical-pairs P R
proof -
  from mgu-vd-complete [OF unif]
  obtain μ1 μ2 where mgu: mgu-vd ren l'' l' = Some (μ1, μ2) by blast
  show ?thesis
  by (intro exI, rule critical-pairsI [OF P R l l'' mgu refl refl b])
qed

```

end
end

6 Preliminaries

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

theory *Preliminaries*

imports

Polynomial-Factorization.Missing-List

begin

lemma *in-set-idx*: $a \in \text{set } as \implies \exists i. i < \text{length } as \wedge a = as ! i$
unfolding *set-conv-nth* **by** *auto*

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)*

using $\langle \text{card } (f \text{ ` } A) = \text{card } A \rangle$

unfolding *inj-on-iff-eq-card* [*OF* $\langle \text{finite } A \rangle$, *symmetric*]

by (*rule inj-on-imp-bij-betw*)

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 $\exists g. \text{finite } \{x. g \ x \neq x\} \wedge$

$(\forall x \in \text{UNIV} - (A \cup B). g \ x = x) \wedge$

$(\forall x \in A. g \ x = f \ x) \wedge$

bij-betw g S S

proof –

have *finite B* **using** *assms* **by** (*metis bij-betw-finite*)

have [*simp*]: $\text{card } A = \text{card } B$ **by** (*metis * bij-betw-same-card*)

have $\text{card } (A - B) = \text{card } (B - A)$

proof –

have $\text{card } (A - B) = \text{card } A - \text{card } (A \cap B)$

by (*metis* $\langle \text{finite } A \rangle$ *card-Diff-subset-Int finite-Int*)

moreover **have** $\text{card } (B - A) = \text{card } B - \text{card } (A \cap B)$

by (*metis* $\langle \text{finite } A \rangle$ *card-Diff-subset-Int finite-Int inf-commute*)

ultimately show *?thesis* **by** *simp*

qed

then obtain *g* **where** **: *bij-betw g (B - A) (A - B)*

by (*metis* $\langle \text{finite } A \rangle$ $\langle \text{finite } B \rangle$ *bij-betw-iff-card finite-Diff*)

define *h* **where** $h = (\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else if } x \in B - A \text{ then } g \ x \text{ else } x)$

```

have bij-betw h A B
  by (metis (full-types) * bij-betw-cong h-def)
moreover have bij-betw h (S - (A ∪ B)) (S - (A ∪ B))
  by (auto simp: bij-betw-def h-def inj-on-def)
moreover have B ∩ (S - (A ∪ B)) = {} by blast
ultimately have bij-betw h (A ∪ (S - (A ∪ B))) (B ∪ (S - (A ∪ B)))
  by (rule bij-betw-combine)
moreover have A ∪ (S - (A ∪ B)) = S - (B - A)
  and B ∪ (S - (A ∪ B)) = S - (A - B)
  using ⟨A ⊆ S⟩ and ⟨B ⊆ S⟩ by blast+
ultimately have bij-betw h (S - (B - A)) (S - (A - B)) by simp
moreover have bij-betw h (B - A) (A - B)
  using ** by (auto simp: bij-betw-def h-def inj-on-def)
moreover have (S - (A - B)) ∩ (A - B) = {} by blast
ultimately have bij-betw h ((S - (B - A)) ∪ (B - A)) ((S - (A - B)) ∪ (A
- B))
  by (rule bij-betw-combine)
moreover have (S - (B - A)) ∪ (B - A) = S
  and (S - (A - B)) ∪ (A - B) = S
  using ⟨A ⊆ S⟩ and ⟨B ⊆ S⟩ by auto
ultimately have bij-betw h S S by simp
moreover have ∀ x ∈ A. h x = f x by (auto simp: h-def)
moreover have finite {x. h x ≠ x}
proof -
  have finite (A ∪ (B - A)) using ⟨finite A⟩ and ⟨finite B⟩ by auto
  moreover have {x. h x ≠ x} ⊆ (A ∪ (B - A)) by (auto simp: h-def)
  ultimately show ?thesis by (metis finite-subset)
qed
moreover have ∀ x ∈ UNIV - (A ∪ B). h x = x by (simp add: h-def)
ultimately show ?thesis by blast
qed

lemma concat-nth:
  assumes m < length xs and n < length (xs ! m)
  and i = sum-list (map length (take m xs)) + n
  shows concat xs ! i = xs ! m ! n
using assms
proof (induct xs arbitrary: m n i)
  case (Cons x xs)
  show ?case
  proof (cases m)
    case 0
    then show ?thesis using Cons by (simp add: nth-append)
  next
    case (Suc k)
    with Cons(1) [of k n i - length x] and Cons(2-)
    show ?thesis by (simp-all add: nth-append)
  qed
qed
qed simp

```

```

lemma concat-nth-length:
   $i < \text{length } \text{uss} \implies j < \text{length } (\text{uss} ! i) \implies$ 
   $\text{sum-list } (\text{map length } (\text{take } i \text{ uss})) + j < \text{length } (\text{concat uss})$ 
proof (induct uss arbitrary: i j)
  case (Cons u uss i j)
  thus ?case by (cases i, auto)
qed auto

lemma less-length-concat:
  assumes  $i < \text{length } (\text{concat } xs)$ 
  shows  $\exists m n.$ 
   $i = \text{sum-list } (\text{map length } (\text{take } m xs)) + n \wedge$ 
   $m < \text{length } xs \wedge n < \text{length } (xs ! m) \wedge \text{concat } xs ! i = xs ! m ! n$ 
using assms
proof (induct xs arbitrary: i rule: length-induct)
  case (1 xs)
  then show ?case
  proof (cases xs)
  case (Cons y ys)
  note [simp] = this
  { assume *:  $i < \text{length } y$ 
    with 1 obtain n where  $i = n$  and  $n < \text{length } y$ 
    and  $y ! i = y ! n$  by simp
    then have  $i = \text{sum-list } (\text{map length } (\text{take } 0 xs)) + n$ 
    and  $0 < \text{length } xs$  and  $n < \text{length } (xs ! 0)$ 
    and  $\text{concat } xs ! i = xs ! 0 ! n$ 
    using * by (auto simp: nth-append)
    then have ?thesis by blast }
  moreover
  { assume *:  $i \geq \text{length } y$ 
    define j where  $j = i - \text{length } y$ 
    then have  $\text{length } ys < \text{length } xs \wedge j < \text{length } (\text{concat } ys)$ 
    using * and 1.prem1 by auto
    with 1 obtain m n where  $j = \text{sum-list } (\text{map length } (\text{take } m ys)) + n$ 
    and  $m < \text{length } ys$  and  $n < \text{length } (ys ! m)$ 
    and  $\text{concat } ys ! j = ys ! m ! n$  by blast
    then have  $i = \text{sum-list } (\text{map length } (\text{take } (\text{Suc } m) xs)) + n$ 
    and  $\text{Suc } m < \text{length } xs$  and  $n < \text{length } (xs ! \text{Suc } m)$ 
    and  $\text{concat } xs ! i = xs ! \text{Suc } m ! n$ 
    using * by (simp-all add: j-def nth-append)
    then have ?thesis by blast }
  ultimately show ?thesis by force
qed simp
qed

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  $\text{concat } (\text{take } i \text{ sss} @ \text{remove-nth } j \text{ (sss ! } i) \# \text{drop } (\text{Suc } i) \text{ sss}) = \text{remove-nth}$ 
 $k \text{ (concat sss)}$ 
using assms
unfolding remove-nth-def
proof (induct sss rule: List.rev-induct)
  case Nil then show ?case by auto
next
  case (snoc ss sss)
  then have  $i = \text{length } sss \vee i < \text{length } sss$  by auto
  then show ?case
  proof
    assume  $i : i = \text{length } sss$ 
    have  $\text{sum-list } (\text{map length } sss) = \text{length } (\text{concat } sss)$  by (simp add: length-concat)
    with snoc i show ?thesis by simp
  next
    assume  $i : i < \text{length } sss$ 
    then have  $\text{nth} : (\text{sss} @ [\text{ss}]) ! i = \text{sss} ! i$  by (simp add: nth-append)
    from  $i$  have  $\text{drop} : \text{drop } (\text{Suc } i) (\text{sss} @ [\text{ss}]) = \text{drop } (\text{Suc } i) \text{ sss} @ [\text{ss}]$  by auto
    from  $i$  have  $\text{take} : \text{take } i (\text{sss} @ [\text{ss}]) = \text{take } i \text{ sss}$  by auto
    from snoc(1)[OF i] snoc(2-) have  $1 : \text{concat } (\text{take } i (\text{sss} @ [\text{ss}]) @$ 
       $(\text{take } j ((\text{sss} @ [\text{ss}]) ! i) @ \text{drop } (\text{Suc } j) ((\text{sss} @ [\text{ss}]) ! i)) \# \text{drop } (\text{Suc } i) (\text{sss}$ 
       $@ [\text{ss}])) =$ 
       $\text{take } k (\text{concat } sss) @ \text{drop } (\text{Suc } k) (\text{concat } sss) @ ss$  unfolding take nth drop
by simp
    from snoc(4) take have  $k : k = \text{sum-list } (\text{map length } (\text{take } i \text{ sss})) + j$  by auto
    from nth snoc(3) have  $j : j < \text{length } (\text{sss} ! i)$  by auto
    have  $\text{takek} : \text{take } (\text{sum-list } (\text{map length } (\text{take } i \text{ sss})) + j) (\text{concat } (\text{sss} @ [\text{ss}])) =$ 
       $\text{take } (\text{sum-list } (\text{map length } (\text{take } i \text{ sss})) + j) (\text{concat } sss)$ 
    using concat-nth-length[OF i j] by auto
    have  $\text{dropk} : \text{drop } (\text{Suc } (\text{sum-list } (\text{map length } (\text{take } i \text{ sss})) + j)) (\text{concat } sss) @$ 
 $ss =$ 
       $\text{drop } (\text{Suc } (\text{sum-list } (\text{map length } (\text{take } i \text{ sss})) + j)) (\text{concat } (\text{sss} @ [\text{ss}]))$ 
    using concat-nth-length[OF i j] by auto
    have  $\text{take } k (\text{concat } sss) @ \text{drop } (\text{Suc } k) (\text{concat } sss) @ ss =$ 
       $\text{take } k (\text{concat } (\text{sss} @ [\text{ss}])) @ \text{drop } (\text{Suc } k) (\text{concat } (\text{sss} @ [\text{ss}]))$ 
    unfolding  $k$  takek dropk ..
    with 1 show ?thesis by auto
  qed
qed

lemma nth-append-Cons:  $(xs @ y \# zs) ! i =$ 
   $(\text{if } i < \text{length } xs \text{ then } xs ! i \text{ else if } i = \text{length } xs \text{ then } y \text{ else } zs ! (i - \text{Suc } (\text{length}$ 
 $xs)))$ 
by (cases i length xs rule: linorder-cases, auto simp: nth-append)

lemma finite-imp-eq [simp]:
   $\text{finite } \{x. P x \longrightarrow Q x\} \longleftrightarrow \text{finite } \{x. \neg P x\} \wedge \text{finite } \{x. Q x\}$ 

```

by (*auto simp: Collect-imp-eq Collect-neg-eq*)

lemma *sum-list-take-eq*:

fixes *xs* :: *nat list*

shows $k < i \implies i < \text{length } xs \implies \text{sum-list } (\text{take } i \text{ } xs) =$
 $\text{sum-list } (\text{take } k \text{ } xs) + xs ! k + \text{sum-list } (\text{take } (i - \text{Suc } k) \text{ } (\text{drop } (\text{Suc } k) \text{ } xs))$
by (*subst id-take-nth-drop [of k] (auto simp: min-def drop-take)*)

lemma *nth-equalityE*:

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

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

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

lemma *fold-map-cong* [*fundef-cong*]:

assumes $a = b$ **and** $xs = ys$
and $\bigwedge x. x \in \text{set } xs \implies f \text{ } x = g \text{ } x$
shows $\text{fold-map } f \text{ } xs \text{ } a = \text{fold-map } g \text{ } ys \text{ } b$
using *assms* **by** (*induct ys arbitrary: a b xs*) *simp-all*

lemma *fold-map-map-conv*:

assumes $\bigwedge x \text{ } ys. x \in \text{set } xs \implies f \text{ } (g \text{ } x) \text{ } (g' \text{ } x @ ys) = (h \text{ } x, ys)$
shows $\text{fold-map } f \text{ } (\text{map } g \text{ } xs) \text{ } (\text{concat } (\text{map } g' \text{ } xs) @ ys) = (\text{map } h \text{ } xs, ys)$
using *assms* **by** (*induct xs*) *simp-all*

lemma *map-fst-fold-map*:

$\text{map } f \text{ } (\text{fst } (\text{fold-map } g \text{ } xs \text{ } y)) = \text{fst } (\text{fold-map } (\lambda a \text{ } b. \text{apfst } f \text{ } (g \text{ } a \text{ } b)) \text{ } xs \text{ } y)$
by (*induct xs arbitrary: y*) (*auto split: prod.splits, metis fst-conv*)

6.1 Combinators

definition *const* :: $'a \Rightarrow 'b \Rightarrow 'a$ **where**

const $\equiv (\lambda x \text{ } y. x)$

definition *flip* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c)$ **where**

flip *f* $\equiv (\lambda x \text{ } y. f \text{ } y \text{ } x)$

declare *flip-def*[*simp*]

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

by (*simp add: const-def*)

lemma *foldr-Cons-append-conv* [*simp*]:

$\text{foldr } (\#) \text{ } xs \text{ } ys = xs @ ys$
by (*induct xs*) *simp-all*

lemma *foldl-flip-Cons*[simp]:
 foldl (flip (#)) xs ys = rev ys @ xs
 by (induct ys arbitrary: xs) simp-all

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

lemma *foldr-flip-rev*[simp]:
 foldr (flip f) (rev xs) a = foldl f a xs
 by (simp add: foldr-conv-foldl)

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

lemma *foldl-flip-rev*[simp]:
 foldl (flip f) a (rev xs) = foldr f xs a
 by (simp add: foldl-conv-foldr)

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

6.2 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
 using *assms* **proof**(induct xs)
 case (Cons x xs)
 then have *IH:distinct* xs
 by (metis dual-order.trans filter.simps(2) impossible-Cons nle-le)
 from Cons(2) have $\text{length } (\text{filter } ((=) x) xs) = 0$
 by (metis (mono-tags) One-nat-def add.right-neutral add-Suc-right filter.simps(2)
 le-less length-0-conv less-Suc0 list.simps(3) list.size(4) nat.inject)
 then have $x \notin \text{set } (xs)$
 by (metis (full-types) filter-empty-conv length-0-conv)
 with *IH* show ?case
 by simp
qed simp

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)
 using *assms* **proof**(induct xs)
 case (Cons x xs)
 {fix i j assume $i < \text{length } xs \wedge j < \text{length } xs \wedge i \neq j \wedge f(xs!i) \wedge f(xs!j)$
 with Cons(2) have $xs!i \neq xs!j$
 by (metis not-less-eq nth-Cons-Suc Suc-inject length-Cons)
 }
 with Cons(1) have *IH:distinct* (filter f xs)

```

    by presburger
show ?case proof(cases f x)
  case True
    with Cons(2) have  $\forall j < \text{length } xs. f (xs ! j) \longrightarrow x \neq xs ! j$  by fastforce
    then have  $x \notin \text{set } (\text{filter } f \text{ } xs)$  by (metis filter-set in-set-conv-nth member-filter)
    then show ?thesis unfolding filter.simps using True IH by simp
  next
    case False
    then show ?thesis unfolding filter.simps using IH by presburger
qed
qed simp

lemma distinct-is-partition:
  assumes distinct xs
  shows is-partition (map ( $\lambda x. \{x\}$ ) xs)
  using assms proof(induct xs)
  case (Cons x xs)
  then show ?case unfolding list.map(2) is-partition-Cons by force
qed (simp add: is-partition-Nil)

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)
  by (smt (verit, del-insts) add-diff-inverse-nat assms(1) assms(2) assms(3) disjoint-iff is-partition-alt is-partition-alt-def length-append mem-simps(9) nat-add-left-cancel-less
nth-append nth-mem)

lemma distinct-is-partition-sets:
  assumes distinct xs
  and  $xs = \text{concat } ys$ 
  shows is-partition (map set ys)
  using assms proof(induct ys arbitrary:xs)
  case (Cons y ys)
  have is-partition (map set ys) proof–
    from Cons(2,3) have distinct (concat ys)
    unfolding concat.simps by simp
    with Cons(1) show ?thesis by simp
  qed
  moreover from Cons(2,3) have  $\text{set } y \cap \bigcup (\text{set } (\text{map set } ys)) = \{\}$ 
  using distinct-append[of y concat ys]by simp
  ultimately show ?case
  unfolding is-partition-Cons list.map by simp
qed (simp add: is-partition-Nil)

end

theory SubList
imports

```

```

HOL-Library.Sublist
HOL-Library.Multiset
begin

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

lemma subseq-Cons-Cons:
  assumes subseq (a # as) (b # bs)
  shows subseq as bs
  using assms by (cases a = b) (auto intro: subseq-Cons')

lemma subseq-induct2:
  [| subseq xs ys;
    ∧ bs. P [| bs;
    ∧ a as bs. [| subseq as bs; P as bs |] ⇒ P (a # as) (a # bs);
    ∧ a as b bs. [| a ≠ b; subseq as bs; subseq (a # as) bs; P as bs; P (a # as) bs |]
    ⇒ P (a # as) (b # bs) |]
  ⇒ P xs ys
proof (induct ys arbitrary: xs)
  case Nil then show ?case by (metis list-emb-Nil2)
next
  case (Cons y ys')
  note Cons-ys = Cons
  note sl = Cons(2)
  note step-eq = Cons(4)
  note step-neq = Cons(5)
  show ?case proof (cases xs)
    case Nil show ?thesis unfolding Nil using Cons.prem(2) by auto
  next
    case (Cons x xs')
    have sl': subseq xs' ys' by (metis Cons sl subseq-Cons-Cons)
    from sl' have P': P xs' ys' using Cons-ys by auto
    show ?thesis proof (cases x = y)
      case False
      have sl'': subseq (x # xs') ys' using sl unfolding Cons using False by auto
      then have P'': P (x # xs') ys' by (metis Cons.hyps Cons-ys(3) step-eq
step-neq)
      show ?thesis using step-neq[OF False sl' sl'' P' P''] unfolding Cons by
auto
    next
      case True
      show ?thesis using step-eq[OF sl' P'] unfolding Cons True[symmetric] by
auto
    qed
  qed
qed
qed

lemma subseq-submultiset:
  subseq xs ys ⇒ mset xs ⊆# mset ys

```

```

    by (induct rule: list-emb.induct) (auto intro: subset-mset.order-trans)

lemma subseq-subset:
  subseq xs ys  $\implies$  set xs  $\subseteq$  set ys
  by (induct rule: list-emb.induct) auto

lemma remove1-subseq:
  subseq (remove1 x xs) xs
  by (induct xs) auto

lemma subseq-concat:
  assumes  $\bigwedge x. x \in \text{set } xs \implies \text{subseq } (f\ x) (g\ x)$ 
  shows subseq (concat (map f xs)) (concat (map g xs))
  using assms by (induct xs) (auto intro: list-emb-append-mono)

end

```

7 Multihole Contexts

```

theory Multihole-Context
imports
  Trs
  Preliminaries
  SubList
begin

unbundle lattice-syntax

```

7.1 Partitioning lists into chunks of given length

```

fun partition-by
where
  partition-by xs [] = [] |
  partition-by xs (y#ys) = take y xs # partition-by (drop y xs) ys

lemma partition-by-map0-append [simp]:
  partition-by xs (map ( $\lambda x. 0$ ) ys @ zs) = replicate (length ys) [] @ partition-by xs
  zs
  by (induct ys) simp-all

lemma concat-partition-by [simp]:
  sum-list ys = length xs  $\implies$  concat (partition-by xs ys) = xs
  by (induct ys arbitrary: xs) simp-all

definition partition-by-idx where
  partition-by-idx l ys i j = partition-by [0..i < \text{length } (\text{partition-by } xs\ ys)

```

and $j < \text{length } (\text{partition-by } xs \text{ } ys ! i)$
and $\text{sum-list } ys = \text{length } xs$
shows $\text{partition-by } xs \text{ } ys ! i ! j = xs ! (\text{sum-list } (\text{map length } (\text{take } i (\text{partition-by } xs \text{ } ys))) + j)$
using $\text{concat-nth } [OF \text{ } \text{assms}(1, 2) \text{ } refl]$
unfolding $\text{concat-partition-by } [OF \text{ } \text{assms}(3)]$ **by** simp

lemma $\text{map-map-partition-by}$:
 $\text{map } (\text{map } f) (\text{partition-by } xs \text{ } ys) = \text{partition-by } (\text{map } f \text{ } xs) \text{ } ys$
by $(\text{induct } ys \text{ } \text{arbitrary: } xs) (\text{auto } \text{simp: } \text{take-map drop-map})$

lemma $\text{length-partition-by } [simp]$:
 $\text{length } (\text{partition-by } xs \text{ } ys) = \text{length } ys$
by $(\text{induct } ys \text{ } \text{arbitrary: } xs) \text{ } \text{simp-all}$

lemma $\text{partition-by-Nil } [simp]$:
 $\text{partition-by } [] \text{ } ys = \text{replicate } (\text{length } ys) []$
by $(\text{induct } ys) \text{ } \text{simp-all}$

lemma $\text{partition-by-concat-id } [simp]$:
assumes $\text{length } xss = \text{length } ys$
and $\bigwedge i. i < \text{length } ys \implies \text{length } (xss ! i) = ys ! i$
shows $\text{partition-by } (\text{concat } xss) \text{ } ys = xss$
using assms
proof $(\text{induct } ys \text{ } \text{arbitrary: } xss)$
case $(\text{Cons } y \text{ } ys \text{ } xss)$
then show $?case$ **by** $(\text{cases } xss; \text{fastforce})$
qed simp

lemma partition-by-nth :
 $i < \text{length } ys \implies \text{partition-by } xs \text{ } ys ! i = \text{take } (ys ! i) (\text{drop } (\text{sum-list } (\text{take } i \text{ } ys)) \text{ } xs)$
proof $(\text{induct } ys \text{ } \text{arbitrary: } xs \text{ } i)$
case $(\text{Cons } x \text{ } xs \text{ } i)$
thus $?case$ **by** $(\text{cases } i, \text{auto } \text{simp: } \text{ac-simps})$
qed simp

lemma $\text{partition-by-nth-less}$:
assumes $k < i$ **and** $i < \text{length } zs$
and $\text{length } xs = \text{sum-list } (\text{take } i \text{ } zs) + j$
shows $\text{partition-by } (xs @ y \# ys) \text{ } zs ! k = \text{take } (zs ! k) (\text{drop } (\text{sum-list } (\text{take } k \text{ } zs)) \text{ } xs)$
proof –
have $\text{partition-by } (xs @ y \# ys) \text{ } zs ! k =$
 $\text{take } (zs ! k) (\text{drop } (\text{sum-list } (\text{take } k \text{ } zs)) \text{ } (xs @ y \# ys))$
using assms **by** $(\text{auto } \text{simp: } \text{partition-by-nth})$
moreover have $zs ! k + \text{sum-list } (\text{take } k \text{ } zs) \leq \text{length } xs$
using assms **by** $(\text{simp add: } \text{sum-list-take-eq})$

ultimately show *?thesis* by *simp*
qed

lemma *partition-by-nth-greater*:

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

proof –

have $\text{partition-by } (xs @ y \# ys) \text{ } zs ! k =$
 $\text{take } (zs ! k) (\text{drop } (\text{sum-list } (\text{take } k \text{ } zs)) (xs @ y \# ys))$
using *assms* by (auto *simp*: *partition-by-nth*)
moreover have $\text{sum-list } (\text{take } k \text{ } zs) > \text{length } xs$
using *assms* by (auto *simp*: *sum-list-take-eq*)
ultimately show *?thesis* by (auto) (*metis* *Suc-diff-Suc drop-Suc-Cons*)

qed

lemma *length-partition-by-nth*:

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

proof (*induct* *ys* arbitrary: *xs* *i*)

case (*Cons* *y* *ys* *xs* *i*)
thus *?case* by (*cases* *i*, *auto*)

qed *simp*

lemma *partition-by-nth-nth-elem*:

assumes $\text{sum-list } ys = \text{length } xs$ $i < \text{length } ys$ $j < ys ! i$
shows $\text{partition-by } xs \text{ } ys ! i ! j \in \text{set } xs$

proof –

from *assms* have $j < \text{length } (\text{partition-by } xs \text{ } ys ! i)$ by (*simp* only: *length-partition-by-nth*)
then have $\text{partition-by } xs \text{ } ys ! i ! j \in \text{set } (\text{partition-by } xs \text{ } ys ! i)$ by *auto*
with *assms*(2) have $\text{partition-by } xs \text{ } ys ! i ! j \in \text{set } (\text{concat } (\text{partition-by } xs \text{ } ys))$

by *auto*

then show *?thesis* using *assms* by *simp*

qed

lemma *partition-by-nth-nth*:

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

unfolding *partition-by-idx-def*

proof –

let *?n* = $\text{partition-by } [0..<\text{length } xs] \text{ } ys ! i ! j$

show *?n* < $\text{length } xs$

using *partition-by-nth-nth-elem*[*OF* - *assms*(2,3), of $[0..<\text{length } xs]$] *assms*(1)

by *simp*

have *li*: $i < \text{length } (\text{partition-by } [0..<\text{length } xs] \text{ } ys)$ using *assms*(2) by *simp*

have *lj*: $j < \text{length } (\text{partition-by } [0..<\text{length } xs] \text{ } ys ! i)$

using *assms* by (*simp* add: *length-partition-by-nth*)

have *partition-by* (map (!) *xs*) [*0*..*length xs*] *ys* ! *i* ! *j* = *xs* ! ?*n*
by (*simp only*: *map-map-partition-by*[*symmetric*] *nth-map*[*OF li*] *nth-map*[*OF lj*])
then show *partition-by xs ys* ! *i* ! *j* = *xs* ! ?*n* **by** (*simp add*: *map-nth*)
qed

lemma *map-length-partition-by* [*simp*]:
 $\text{sum-list } ys = \text{length } xs \implies \text{map length } (\text{partition-by } xs \text{ } ys) = ys$
by (*intro nth-equalityI*, *auto simp*: *length-partition-by-nth*)

lemma *map-partition-by-nth* [*simp*]:
 $i < \text{length } ys \implies \text{map } f (\text{partition-by } xs \text{ } ys ! i) = \text{partition-by } (\text{map } f \text{ } xs) \text{ } ys ! i$
proof (*induct ys arbitrary: i xs*)
case (*Cons y ys i xs*)
thus ?*case* **by** (*cases i*, *simp-all add*: *take-map drop-map*)
qed simp

lemma *sum-list-partition-by* [*simp*]:
 $\text{sum-list } ys = \text{length } xs \implies$
 $\text{sum-list } (\text{map } (\lambda x. \text{sum-list } (\text{map } f \text{ } x)) (\text{partition-by } xs \text{ } ys)) = \text{sum-list } (\text{map } f \text{ } xs)$
by (*induct ys arbitrary: xs*) (*simp-all*, *metis append-take-drop-id sum-list-append map-append*)

lemma *partition-by-map-conv*:
 $\text{partition-by } xs \text{ } ys = \text{map } (\lambda i. \text{take } (ys ! i) (\text{drop } (\text{sum-list } (\text{take } i \text{ } ys)) \text{ } xs)) [0 .. < \text{length } ys]$
by (*rule nth-equalityI*) (*simp-all add*: *partition-by-nth*)

lemma *UN-set-partition-by-map*:
 $\text{sum-list } ys = \text{length } xs \implies (\bigcup x \in \text{set } (\text{partition-by } (\text{map } f \text{ } xs) \text{ } ys). \bigcup (set \text{ } x)) = \bigcup (set \text{ } (\text{map } f \text{ } xs))$
by (*induct ys arbitrary: xs*)
(simp-all add: *drop-map take-map*, *metis UN-Un append-take-drop-id set-append*)

lemma *UN-set-partition-by*:
 $\text{sum-list } ys = \text{length } xs \implies (\bigcup zs \in \text{set } (\text{partition-by } xs \text{ } ys). \bigcup x \in \text{set } zs. f \text{ } x) = (\bigcup x \in \text{set } xs. f \text{ } x)$
by (*induct ys arbitrary: xs*) (*simp-all*, *metis UN-Un append-take-drop-id set-append*)

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

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

```

case (Cons y ys)
then show ?case
  apply (subst (2) append-take-drop-id [of y xs, symmetric])
  apply (simp only: set-append)
  apply auto
done
qed simp

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
by (induct ys arbitrary: xs) (auto simp: drop-take ac-simps split: split-min)

lemma partition-by-concat2:
  partition-by xs (concat ys) =
    concat (map ( $\lambda i$  . partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i))
[0..length ys])
proof -
  have *: map ( $\lambda i$  . partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i))
[0..length ys] =
  map ( $\lambda(x,y)$ . partition-by x y) (zip (partition-by xs (map sum-list ys)) ys)
  using zip-nth-conv[of partition-by xs (map sum-list ys) ys] by auto
  show ?thesis unfolding * by (induct ys arbitrary: xs) (auto simp: partition-by-append2)
qed

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 ( $\lambda i$ . partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i)) [0..length
ys]
by (auto simp: partition-by-concat2 intro: partition-by-concat-id)

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

## 7.2 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

```

```

by (induct t) (simp-all add: map-idI)

fun num-holes :: ('f, 'v) mctxt  $\Rightarrow$  nat
where
  num-holes (MVar -) = 0 |
  num-holes MHole = 1 |
  num-holes (MFun - ctxts) = sum-list (map num-holes ctxts)

lemma num-holes-o-mctxt-of-term [simp]:
  num-holes  $\circ$  mctxt-of-term = ( $\lambda x$ . 0)
apply (intro ext)
subgoal for x by (induct x, auto)
done

lemma mctxt-of-term-term-of-mctxt-id [simp]:
  num-holes C = 0  $\implies$  mctxt-of-term (term-of-mctxt C) = C
by (induct C) (simp-all add: map-idI)

lemma vars-mctxt-of-term [simp]: vars-mctxt (mctxt-of-term t) = vars-term t
by (induct t, auto)

lemma num-holes-mctxt-of-term [simp]:
  num-holes (mctxt-of-term t) = 0
by (induct t) simp-all

fun funas-mctxt :: ('f, 'v) mctxt  $\Rightarrow$  'f sig
where
  funas-mctxt (MFun f Cs) = {(f, length Cs)}  $\cup \bigcup$  (funas-mctxt ' set Cs) |
  funas-mctxt - = {}

fun funas-mctxt-list :: ('f, 'v) mctxt  $\Rightarrow$  ('f  $\times$  nat) list
where
  funas-mctxt-list (MFun f Cs) = (f, length Cs) # concat (map funas-mctxt-list Cs) |
  funas-mctxt-list - = []

lemma funas-mctxt-list [simp]:
  set (funas-mctxt-list C) = funas-mctxt C
by (induct C) simp-all

fun split-term :: (('f, 'v) term  $\Rightarrow$  bool)  $\Rightarrow$  ('f, 'v) term  $\Rightarrow$  ('f, 'v) mctxt  $\times$  ('f, 'v)
term list
where
  split-term P (Var x) = (if P (Var x) then (MHole, [Var x]) else (MVar x, [])) |
  split-term P (Fun f ts) =
    (if P (Fun f ts) then (MHole, [Fun f ts])
     else let us = map (split-term P) ts in (MFun f (map fst us), concat (map snd us)))

```

fun *cap-till* :: ((*f*, '*v*') *term* \Rightarrow *bool*) \Rightarrow (*f*, '*v*') *term* \Rightarrow (*f*, '*v*') *mctxt*
where
 cap-till *P* (*Var* *x*) = (if *P* (*Var* *x*) then *MHole* else *MVar* *x*) |
 cap-till *P* (*Fun* *f* *ts*) = (if *P* (*Fun* *f* *ts*) then *MHole* else *MFun* *f* (map (*cap-till* *P*) *ts*))

fun *uncap-till* :: ((*f*, '*v*') *term* \Rightarrow *bool*) \Rightarrow (*f*, '*v*') *term* \Rightarrow (*f*, '*v*') *term list*
where
 uncap-till *P* (*Var* *x*) = (if *P* (*Var* *x*) then [*Var* *x*] else []) |
 uncap-till *P* (*Fun* *f* *ts*) = (if *P* (*Fun* *f* *ts*) then [*Fun* *f* *ts*] else concat (map (*uncap-till* *P*) *ts*))

lemma *split-term* [*simp*]:
 split-term *P* *t* = (*cap-till* *P* *t*, *uncap-till* *P* *t*)
by (*induct* *t*) (*simp-all cong: map-cong*)

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

lemma *if-Fun-in-set-simps* [*simp*]:
 if-Fun-in-set *F* (*Var* *x*)
 if-Fun-in-set *F* (*Fun* *f* *ts*) $\longleftrightarrow (f, \text{length } ts) \in F$
 is-Var *t* \Longrightarrow *if-Fun-in-set* *F* *t*
 is-Fun *t* \Longrightarrow *if-Fun-in-set* *F* *t* $\longleftrightarrow \text{the } (\text{root } t) \in F$
by (*simp-all add: if-Fun-in-set-def*)

lemma *if-Fun-in-set-mono*:
 F \subseteq *G* \Longrightarrow *if-Fun-in-set* *F* *t* \Longrightarrow *if-Fun-in-set* *G* *t*
by (*auto simp: if-Fun-in-set-def*)

abbreviation *split-term-funas* *F* \equiv *split-term* (*if-Fun-in-set* *F*)

abbreviation *cap-till-funas* *F* \equiv *cap-till* (*if-Fun-in-set* *F*)

abbreviation *uncap-till-funas* *F* \equiv *uncap-till* (*if-Fun-in-set* *F*)

lemma *if-Fun-in-set-uncap-till-funas*:
 A \subseteq *B* \Longrightarrow *if-Fun-in-set* *A* *t* \Longrightarrow *uncap-till-funas* *B* *t* = [*t*]
by (*cases* *t*) *auto*

lemma *cap-till-funasD* [*dest*]:
 fn \in *funas-mctxt* (*cap-till-funas* *F* *t*) \Longrightarrow *fn* \in *F* \Longrightarrow *False*
proof (*induct* *t*)
 case (*Fun* *f* *ts*)
 then show ?*case* **by** (*cases* (*f*, *length* *ts*) \in *F*) *auto*
qed *simp*

lemma *cap-till-funas*:
 $\forall \text{fn} \in \text{funas-mctxt } (\text{cap-till-funas } F \text{ } t). \text{fn} \notin F$
by *auto*

lemma *uncap-till*:

$\forall s \in \text{set } (\text{uncap-till } P \ t). \ P \ s$
by (*induct t*) *simp-all*

lemma *uncap-till-singleton*:

assumes $s \in \text{set } (\text{uncap-till } P \ t)$
shows $\text{uncap-till } P \ s = [s]$
using *assms*
proof (*induct t*)
case (*Fun f ts*)
then show $?case$ **by** (*cases P (Fun f ts)*) *auto*
qed *simp*

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

$\text{map } (\text{uncap-till } P) (\text{uncap-till } P \ t) = \text{map } (\lambda s. [s]) (\text{uncap-till } P \ t)$
by (*intro map-cong [OF refl] uncap-till-singleton*) *simp-all*

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

$P \ (\text{Fun } f \ ts) \implies \text{uncap-till } P \ (\text{Fun } f \ ts) = [\text{Fun } f \ ts]$
by *simp*

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* |
fill-holes *MHole [t]* = *t* |
fill-holes (*MFun f cs*) *ts* = *Fun f* ($\text{map } (\lambda i. \text{fill-holes } (cs \ ! \ i) \ (\text{partition-holes } ts \ cs \ ! \ i)) \ [0 \ ..< \ \text{length } cs])$)

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* $?ys = \text{length } ?xs \implies \text{concat } (\text{partition-by } ?xs \ ?ys) = ?xs$ 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 *len1*: $\text{num-holes } C = \text{length } xs$ **and** *len2*: $\text{num-holes } C = \text{length } ys$
and *Hole*: $\bigwedge x y. P \ \text{MHole } [x] \ [y]$
and *Var*: $\bigwedge v. P \ (\text{MVar } v) \ \square \ \square$
and *Fun*: $\bigwedge f \ Cs \ xs \ ys. \ \text{sum-list } (\text{map num-holes } Cs) = \text{length } xs \implies$
 $\text{sum-list } (\text{map num-holes } Cs) = \text{length } ys \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P \ (Cs \ ! \ i) \ (\text{partition-holes } xs \ Cs \ ! \ i) \ (\text{partition-holes } ys \ Cs \ ! \ i)) \implies$
 $P \ (\text{MFun } f \ Cs) \ (\text{concat } (\text{partition-holes } xs \ Cs)) \ (\text{concat } (\text{partition-holes } ys \ Cs))$
shows $P \ C \ xs \ ys$
proof (*insert len1 len2, induct C arbitrary: xs ys*)
case *MHole* **then show** $?case$ **using** *Hole* **by** (*cases xs; cases ys*) *auto*

```

next
  case (MVar v) then show ?case using Var by auto
next
  case (MFun f Cs) then show ?case using Fun[of Cs xs ys f] by (auto simp:
length-partition-by-nth)
qed

```

```

lemma fill-holes-induct[consumes 1, case-names MHole MVar MFun]:
  fixes P :: ('f,'v) mtxt  $\Rightarrow$  'a list  $\Rightarrow$  bool
  assumes len: num-holes C = length xs
  and Hole:  $\bigwedge x. P \text{ MHole } [x]$ 
  and Var:  $\bigwedge v. P (MVar v)$ 
  and Fun:  $\bigwedge f \text{ Cs xs. sum-list (map num-holes Cs) = length xs} \Rightarrow$ 
    ( $\bigwedge i. i < \text{length Cs} \Rightarrow P (Cs ! i) (\text{partition-holes xs Cs ! i}) \Rightarrow$ 
      P (MFun f Cs) (concat (partition-holes xs Cs)))
  shows P C xs
using fill-holes-induct2[of C xs xs  $\lambda C \text{ xs. } P C \text{ xs}$ ] assms by simp

```

```

lemma funas-term-fill-holes-iff: num-holes C = length ts  $\Rightarrow$ 
  g  $\in$  funas-term (fill-holes C ts)  $\longleftrightarrow$  g  $\in$  funas-mtxt C  $\vee$  ( $\exists t \in \text{set ts. } g \in$ 
  funas-term t)
proof (induct C ts rule: fill-holes-induct)
  case (MFun f Cs ts)
  have ( $\exists i < \text{length Cs. } g \in \text{funas-term (fill-holes (Cs ! i) (\text{partition-holes (concat}$ 
  (partition-holes ts Cs)) Cs ! i)))
   $\longleftrightarrow$  ( $\exists C \in \text{set Cs. } g \in \text{funas-mtxt C}$ )  $\vee$  ( $\exists us \in \text{set (partition-holes ts Cs).}$ 
 $\exists t \in \text{set us. } g \in \text{funas-term t}$ )
  using MFun by (auto simp: ex-set-conv-ex-nth)
  then show ?case by auto
qed auto

```

```

lemma fill-holes-MHole:
  length ts = 1  $\Rightarrow$  ts ! 0 = u  $\Rightarrow$  fill-holes MHole ts = u
  by (cases ts) simp-all

```

```

lemmas
  map-partition-holes-nth [simp] =
  map-partition-by-nth [of - map num-holes Cs for Cs, unfolded length-map] and
  length-partition-holes [simp] =
  length-partition-by [of - map num-holes Cs for Cs, unfolded length-map]

```

```

lemma length-partition-holes-nth [simp]:
  assumes sum-list (map num-holes cs) = length ts
  and i < length cs
  shows length (partition-holes ts cs ! i) = num-holes (cs ! i)
  using assms by (simp add: length-partition-by-nth)

```

```

lemma concat-partition-holes-upt:
  assumes  $i \leq \text{length } cs$ 
  shows  $\text{concat } [\text{partition-holes } ts \text{ } cs ! j. j \leftarrow [0 \dots i]] =$ 
     $\text{take } (\text{sum-list } [\text{num-holes } (cs ! j). j \leftarrow [0 \dots i]]) \text{ } ts$ 
using assms
proof (induct i arbitrary: ts)
  case (Suc i)
  then have  $i': i < \text{length } cs$  by (metis less-eq-Suc-le)
  then have  $*$ ;  $i < \text{length } (\text{map num-holes } cs)$  by simp
  then have  $i'': i \leq \text{length } cs$  by auto
  show ?case
    unfolding upt-Suc-append [OF le0] map-append concat-append Suc(1) [OF i'']
concat.simps append-Nil2
    unfolding sum-list-append take-add
    unfolding list.map(2)
    unfolding partition-by-nth [OF *]
    unfolding take-map nth-map [OF i']
    unfolding take-upt-id [OF i']
    unfolding map-map o-def by auto
qed (auto)

```

```

lemma partition-holes-step:
   $\text{partition-holes } ts \text{ } (C \# Cs) = \text{take } (\text{num-holes } C) \text{ } ts \# \text{partition-holes } (\text{drop}$ 
     $(\text{num-holes } C) \text{ } ts) \text{ } Cs$ 
  by simp

```

```

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  $\text{partition-holes } ts \text{ } cs = \text{partition-holes } ts \text{ } ds$ 
  using assms by (metis nth-map-conv)

```

```

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  $\text{partition-holes } (\text{concat } sss) \text{ } cs = sss$ 
  using assms by (intro partition-by-concat-id) auto

```

```

lemma partition-holes-fill-holes-conv:
   $\text{fill-holes } (MFun \text{ } f \text{ } cs) \text{ } ts =$ 
     $Fun \text{ } f \text{ } [\text{fill-holes } (cs ! i) \text{ } (\text{partition-holes } ts \text{ } cs ! i). i \leftarrow [0 \dots \text{length } cs]]$ 
  by (simp add: partition-by-nth take-map)

```

```

lemma fill-holes-arbitrary:
  assumes  $lCs: \text{length } Cs = \text{length } ts$ 

```

and lss : $length\ ss = length\ ts$
and rec : $\bigwedge i. i < length\ ts \implies num_holes\ (Cs\ !\ i) = length\ (ss\ !\ i) \wedge f\ (Cs\ !\ i)\ (ss\ !\ i) = ts\ !\ i$
shows $map\ (\lambda i. f\ (Cs\ !\ i)\ (partition_holes\ (concat\ ss)\ Cs\ !\ i))\ [0 ..< length\ Cs]$
 $= ts$
proof –
have $sum_list\ (map\ num_holes\ Cs) = length\ (concat\ ss)$ **using** $assms$
by $(auto\ simp: length_concat\ map_nth_eq_conv\ intro: arg_cong[of\ -\ -\ sum_list])$
moreover **have** $partition_holes\ (concat\ ss)\ Cs = ss$
using $assms$ **by** $(auto\ intro: partition_by_concat_id)$
ultimately show $?thesis$ **using** $assms$ **by** $(auto\ intro: nth_equalityI)$
qed

lemma $fill_holes_MFun$:
assumes lCs : $length\ Cs = length\ ts$
and lss : $length\ ss = length\ ts$
and rec : $\bigwedge i. i < length\ ts \implies num_holes\ (Cs\ !\ i) = length\ (ss\ !\ i) \wedge fill_holes\ (Cs\ !\ i)\ (ss\ !\ i) = ts\ !\ i$
shows $fill_holes\ (MFun\ f\ Cs)\ (concat\ ss) = Fun\ f\ ts$
unfolding $fill_holes.simps\ term.simps$
by $(rule\ conjI[OF\ refl], rule\ fill_holes_arbitrary[OF\ lCs\ lss\ rec])$

inductive
 $eq_fill ::$
 $(f, 'v)\ term \Rightarrow (f, 'v)\ mctx \times (f, 'v)\ term\ list \Rightarrow bool\ ((-/ =_f -)\ [51, 51]\ 50)$
where
 $eqfI\ [intro]: t = fill_holes\ D\ ss \implies num_holes\ D = length\ ss \implies t =_f\ (D, ss)$

lemma $fill_holes_inj$:
assumes $num_holes\ C = length\ ss$
and $num_holes\ C = length\ ts$
and $fill_holes\ C\ ss = fill_holes\ C\ ts$
shows $ss = ts$
using $assms$
proof $(induct\ C\ ss\ ts\ rule: fill_holes_induct2)$
case $(MFun\ f\ Cs\ ss\ ts)$
then show $?case$ **by** $(intro\ arg_cong[of\ -\ -\ concat]\ nth_equalityI)\ auto$
qed $auto$

lemma $eqf_refl\ [intro]$:
 $num_holes\ C = length\ ts \implies fill_holes\ C\ ts =_f\ (C, ts)$
by $(auto)$

lemma $eqfE$:
assumes $t =_f\ (D, ss)$ **shows** $t = fill_holes\ D\ ss\ num_holes\ D = length\ ss$
using $assms[unfolded\ eq_fill.simps]$ **by** $auto$

lemma eqf_MFunI :
assumes $length\ sss = length\ Cs$

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
have $\text{num-holes } (\text{MFun } f \text{ } Cs) = \text{sum-list } (\text{map num-holes } Cs)$ **by** *simp*
also have $\text{map num-holes } Cs = \text{map length } sss$
by (*rule nth-equalityI*, *insert assms eqfE[OF assms(3)]*, *auto*)
also have $\text{sum-list } (...) = \text{length } (\text{concat } sss)$ **unfolding** *length-concat ..*
finally show $\text{num-holes } (\text{MFun } f \text{ } Cs) = \text{length } (\text{concat } sss)$.
show $\text{Fun } f \text{ } ts = \text{fill-holes } (\text{MFun } f \text{ } Cs) (\text{concat } sss)$
by (*rule fill-holes-MFun[symmetric]*, *insert assms(1,2) eqfE[OF assms(3)]*,
auto)
qed

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 –
from *eqfE[OF assms]* **have** $fh: s = \text{fill-holes } (\text{MFun } f \text{ } Cs) \text{ } ss$
and $nh: \text{sum-list } (\text{map num-holes } Cs) = \text{length } ss$ **by** *auto*
from fh **obtain** ts **where** $s: s = \text{Fun } f \text{ } ts$ **by** (*cases s*, *auto*)
from $fh[\text{unfolded } s]$
have $ts: ts = \text{map } (\lambda i. \text{fill-holes } (Cs ! i) (\text{partition-holes } ss \text{ } Cs ! i)) [0..<\text{length } Cs]$
 $(\text{is } - = \text{map } (?f \text{ } Cs \text{ } ss) \text{ } -)$
by *auto*
let $?sss = \text{partition-holes } ss \text{ } Cs$
from nh
have $*$: $\text{length } ?sss = \text{length } Cs$ $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, ?sss ! i)$
 $ss = \text{concat } ?sss$
by (*auto simp: ts*)
have $len: \text{length } ts = \text{length } Cs$ **unfolding** ts **by** *auto*
assume $ass: \bigwedge ts \text{ } sss. s = \text{Fun } f \text{ } ts \implies$
 $\text{length } ts = \text{length } Cs \implies$
 $\text{length } sss = \text{length } Cs \implies (\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss$
 $! i)) \implies ss = \text{concat } sss \implies \text{thesis}$
show *thesis*
by (*rule ass[OF s len *]*)
qed

lemma *eqf-MHoleE*:

assumes $s =_f (\text{MHole}, ss)$
shows $ss = [s]$
using *assms*
proof (*cases ss*)
case (*Cons x xs*) **with** *assms* **show** *?thesis* **by** (*cases xs*) (*auto dest: eqfE*)
qed (*auto dest: eqfE*)

```

fun mctxt-of-ctxt :: ('f, 'v) ctxt  $\Rightarrow$  ('f, 'v) mctxt
where
  mctxt-of-ctxt Hole = MHole |
  mctxt-of-ctxt (More f ss1 C ss2) =
    MFun f (map mctxt-of-term ss1 @ mctxt-of-ctxt C # map mctxt-of-term ss2)

lemma num-holes-mctxt-of-ctxt [simp]:
  num-holes (mctxt-of-ctxt C) = 1
by (induct C) simp-all

lemma mctxt-of-term: t =f (mctxt-of-term t, [])
proof (induct t)
  case (Var x)
  show ?case by auto
next
  case (Fun f ts)
  let ?ss = map (λ -. []) ts
  have id: concat ?ss = [] by simp
  have ?case = (Fun f ts =f (MFun f (map mctxt-of-term ts), concat ?ss)) un-
folding id by simp
  also have ...
    by (rule eqf-MFunI, insert Fun[unfolded set-conv-nth], auto)
  finally show ?case .
qed

lemma mctxt-of-ctxt [simp]:
  C⟨t⟩ =f (mctxt-of-ctxt C, [t])
proof (induct C)
  case (More f bef C aft)
  let ?sss = map (λ -. []) bef @ [t] # map (λ -. []) aft
  let ?ts = map mctxt-of-term bef @ mctxt-of-ctxt C # map mctxt-of-term aft
  have id: concat ?sss = [t] by (induct bef, auto)
  have ?case =
    (Fun f (bef @ C⟨t⟩ # aft) =f (MFun f ?ts, concat ?sss))
  unfolding id by simp
  also have ...
  proof (rule eqf-MFunI)
    fix i
    assume i: i < length ?ts
    show (bef @ C⟨t⟩ # aft) ! i =f (?ts ! i, ?sss ! i)
    using More i
    by (cases i < length bef, simp add: nth-append mctxt-of-term,
        cases i = length bef, auto simp: nth-append mctxt-of-term)
  qed auto
  finally show ?case .
qed auto

lemma fill-holes-ctxt-main':

```

```

assumes num-holes  $C = \text{Suc } (\text{length } \text{bef} + \text{length } \text{aft})$ 
shows  $\exists D. (\forall s. \text{fill-holes } C (\text{bef } @ s \# \text{aft}) = D \langle s \rangle) \wedge (C = \text{MFun } f \text{ cs} \longrightarrow$ 
 $D \neq \square)$ 
using assms
proof (induct C arbitrary: bef aft)
  case MHole
  show ?case
    by (rule exI[of -  $\square$ ], insert MHole, auto)
next
  case (MFun f cs)
  note  $IH = \text{MFun}(1)$ 
  note  $\text{holes} = \text{MFun}(2)$ 
  let  $?p = \lambda \text{ bef aft b a D cs s. map } (\lambda i. \text{fill-holes } (\text{cs } ! i)$ 
 $(\text{partition-holes } (\text{bef } @ s \# \text{aft}) \text{ cs } ! i)) [0..<\text{length } \text{cs}] =$ 
 $\text{b } @ D \langle s \rangle \# a$ 
  from holes IH
  have  $\exists b D a. \forall s. ?p \text{ bef aft b a D cs s}$ 
  proof (induct cs arbitrary: bef)
    case (Cons c ccs)
    have  $\text{len}: \text{length } (c \# \text{ccs}) = \text{Suc } (\text{length } \text{ccs})$  by simp
    show ?case
    proof (cases num-holes c  $\leq$  length bef)
      case True
      then have  $\text{bef} = \text{take } (\text{num-holes } c) \text{ bef } @ \text{drop } (\text{num-holes } c) \text{ bef}$ 
 $\wedge \text{length } (\text{take } (\text{num-holes } c) \text{ bef}) = \text{num-holes } c$  by auto
      then obtain bc ba where  $\text{bef} = \text{bc } @ \text{ba}$  and  $\text{lbc}: \text{length } \text{bc} = \text{num-holes}$ 
 $c$  by blast
      from Cons(2) have  $\text{nh}: \text{num-holes } (\text{MFun } f \text{ ccs}) = \text{Suc } (\text{length } \text{ba} + \text{length}$ 
 $\text{aft})$  unfolding bef
      by (simp add: lbc)
      from Cons(1)[OF nh Cons(3)] obtain b D a where  $IH: \bigwedge s. ?p \text{ ba aft b a}$ 
 $D \text{ ccs s}$  by auto
      show ?thesis unfolding len map-upt-Suc bef
      by (intro exI[of - fill-holes c bc  $\#$  b] exI[of - D] exI[of - a], insert IH lbc,
 $\text{auto}$ )
    next
    case False
    then have  $\exists la. \text{num-holes } c = \text{Suc } (\text{length } \text{bef} + la)$  by arith
    then obtain la where  $\text{nhc}: \text{num-holes } c = \text{Suc } (\text{length } \text{bef} + la)$  ..
    from Cons(2) nhc have  $\text{length } (\text{take } la \text{ aft}) = la$  by auto
    from Cons(3)[of c bef take la aft, unfolded this, OF - nhc]
    obtain D where  $D: \forall s. \text{fill-holes } c (\text{bef } @ s \# \text{take } la \text{ aft}) = D \langle s \rangle$  by auto
    show ?thesis unfolding len map-upt-Suc
    by (rule exI[of - Nil], rule exI[of - D], simp add: nhc D)
  qed
qed auto
  then obtain b D a where  $\text{main}: \bigwedge s. ?p \text{ bef aft b a D cs s}$  by blast
  show ?case by (rule exI[of - More f b D a], insert main, auto)
qed simp

```

```

lemma fill-holes-ctxt-main:
  assumes num-holes  $C = \text{Suc } (\text{length } \text{bef} + \text{length } \text{aft})$ 
  shows  $\exists D. \forall s. \text{fill-holes } C (\text{bef} @ s \# \text{aft}) = D \langle s \rangle$ 
using assms fill-holes-ctxt-main' by fast

lemma fill-holes-ctxt:
  assumes nh: num-holes  $C = \text{length } ss$ 
  and i:  $i < \text{length } ss$ 
  obtains  $D$  where  $\bigwedge s. \text{fill-holes } C (ss[i := s]) = D \langle s \rangle$ 
proof -
  from id-take-nth-drop[OF i] obtain bef aft where  $ss = \text{bef} @ ss ! i \# \text{aft}$ 
  and bef:  $\text{bef} = \text{take } i \text{ } ss$  by blast
  from bef i have bef:  $\text{length } \text{bef} = i$  by auto
  note len = arg-cong[OF ss, of length]
  from len nh
  have num-holes  $C = \text{Suc } (\text{length } \text{bef} + \text{length } \text{aft})$  by simp
  from fill-holes-ctxt-main[OF this] obtain  $D$  where id:  $\bigwedge s. \text{fill-holes } C (\text{bef} @$ 
 $s \# \text{aft}) = D \langle s \rangle$  by blast
  {
    fix s
    have  $ss[i := s] = \text{bef} @ s \# \text{aft}$  unfolding arg-cong[OF ss, of  $\lambda ss. ss[i := s]$ ]
    using i bef by auto
    with id[of s] have  $\text{fill-holes } C (ss[i := s]) = D \langle s \rangle$  by simp
  }
  then have main:  $\exists D. \forall s. \text{fill-holes } C (ss[i := s]) = D \langle s \rangle$  by blast
  assume  $\bigwedge D. \llbracket \bigwedge s. \text{fill-holes } C (ss[i := s]) = D \langle s \rangle \rrbracket \implies \text{thesis}$ 
  with main show thesis by blast
qed

fun map-vars-mctxt ::  $(v \Rightarrow w) \Rightarrow (f, v) \text{ mctxt} \Rightarrow (f, w) \text{ mctxt}$ 
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  $(\lambda x. x) C = C$ 
by (induct C, auto intro: nth-equalityI)

lemma num-holes-map-vars-mctxt [simp]:
  num-holes (map-vars-mctxt vw C) = num-holes C
proof (induct C)
  case (MFun f Cs)
  then show ?case by (induct Cs, auto)
qed auto

lemma map-vars-term-eq-fill:
   $t =_f (C, ss) \implies \text{map-vars-term } vw \text{ } t =_f (\text{map-vars-mctxt } vw \text{ } C, \text{map } (\text{map-vars-term}$ 

```

```

vw) ss)
proof (induct C arbitrary: t ss)
  case (MFun f Cs s ss)
    from eqf-MFunE[OF MFun(2)] obtain ts sss where s: s = Fun f ts and len:
length ts = length Cs length sss = length Cs
    and IH:  $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$  and ss: ss = concat
sss by metis
    {
      fix i
      assume i: i < length Cs
      then have Cs ! i  $\in$  set Cs by auto
      from MFun(1)[OF this IH[OF i]] have map-vars-term vw (ts ! i) =f (map-vars-mtxt
vw (Cs ! i), map (map-vars-term vw) (sss ! i)) .
    } note IH = this
    show ?case unfolding map-vars-mtxt.simps ss map-concat s term.map
    by (rule eqf-MFunI, insert IH len, auto)
next
  case (MHole t ss)
  from eqfE[OF this]
  show ?case by (cases ss, auto)
next
  case (MVar v t ss)
  from eqfE[OF this]
  show ?case by (cases ss, auto)
qed

```

```

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-mtxt vw C) (map (map-vars-term vw) ss)
proof -
  from eqfE[OF map-vars-term-eq-fill[OF eqfI[OF refl nh]]]
  show ?thesis by simp
qed

```

```

lemma split-term-eqf:
  t =f (cap-till P t, uncap-till P t)
proof (induct t)
  case (Fun f ts)
  show ?case
  proof (cases P (Fun f ts))
  case False
  then have ?thesis = (Fun f ts =f (MFun f (map (cap-till P) ts), concat (map
(uncap-till P) ts)))
  by simp
  also have ...
  proof (rule eqf-MFunI)
  fix i
  presume i < length ts

```

```

    moreover then have  $ts ! i \in \text{set } ts$  by auto
    ultimately show  $ts ! i =_f (\text{map } (\text{cap-till } P) \text{ } ts ! i, \text{map } (\text{uncap-till } P) \text{ } ts ! i)$ 
      using Fun by auto
    qed simp-all
    finally show ?thesis .
  qed auto
qed auto

```

```

lemma fill-holes-cap-till-uncap-till-id [simp]:
   $\text{fill-holes } (\text{cap-till } P \text{ } t) (\text{uncap-till } P \text{ } t) = t$ 
proof -
  have  $t =_f (\text{cap-till } P \text{ } t, \text{uncap-till } P \text{ } t)$  by (metis split-term-egf)
  from egfE [OF this] show ?thesis by simp
qed

```

```

lemma num-holes-cap-till [simp]:
   $\text{num-holes } (\text{cap-till } P \text{ } t) = \text{length } (\text{uncap-till } P \text{ } t)$ 
  using egfE [OF split-term-egf] by auto

```

```

fun split-vars ::  $(f, 'v) \text{ term} \Rightarrow ((f, 'v) \text{ mctxt} \times 'v \text{ list})$ 
where
  split-vars (Var  $x$ ) = (MHole, [ $x$ ]) |
  split-vars (Fun  $f \text{ } ts$ ) = (MFun  $f$  ( $\text{map } (\text{fst} \circ \text{split-vars}) \text{ } ts$ ),  $\text{concat } (\text{map } (\text{snd} \circ \text{split-vars}) \text{ } ts)$ )

```

```

lemma split-vars-num-holes:  $\text{num-holes } (\text{fst } (\text{split-vars } t)) = \text{length } (\text{snd } (\text{split-vars } t))$ 
proof (induct  $t$ )
  case (Fun  $f \text{ } ts$ )
  then show ?case by (induct  $ts$ , auto)
qed simp

```

```

lemma split-vars-vars-term-list:  $\text{snd } (\text{split-vars } t) = \text{vars-term-list } t$ 
proof (induct  $t$ )
  case (Fun  $f \text{ } ts$ )
  then show ?case by (auto simp: vars-term-list.simps o-def, induct  $ts$ , auto)
qed (auto simp: vars-term-list.simps)

```

```

lemma split-vars-vars-term:  $\text{set } (\text{snd } (\text{split-vars } t)) = \text{vars-term } t$ 
  using arg-cong [OF split-vars-vars-term-list [of t], of set] by auto

```

```

lemma split-vars-egf-subst-map-vars-term:
   $t \cdot \sigma =_f (\text{map-vars-mctxt } vw (\text{fst } (\text{split-vars } t)), \text{map } \sigma (\text{snd } (\text{split-vars } t)))$ 
proof (induct  $t$ )
  case (Fun  $f \text{ } ts$ )
  have ?case = (Fun  $f$  ( $\text{map } (\lambda t. t \cdot \sigma) \text{ } ts$ )
    =f (MFun  $f$  ( $\text{map } (\text{map-vars-mctxt } vw \circ (\text{fst} \circ \text{split-vars})) \text{ } ts$ ),  $\text{concat } (\text{map } (\text{map } \sigma \circ (\text{snd} \circ \text{split-vars})) \text{ } ts)$ )))
  by (simp add: map-concat)

```

```

also have ...
proof (rule eqf-MFunI, unfold length-map)
  fix i
  assume i: i < length ts
  then have mem: ts ! i ∈ set ts by auto
  show map (λt. t · σ) ts ! i =f (map (map-vars-mtxt vw ∘ (fst ∘ split-vars))
    ts ! i, map (map σ ∘ (snd ∘ split-vars)) ts ! i)
    using Fun[OF mem] i by auto
  qed auto
  finally show ?case by simp
qed auto

lemma split-vars-eqf-subst: t · σ =f (fst (split-vars t), (map σ (snd (split-vars
t))))
  using split-vars-eqf-subst-map-vars-term[of t σ λ x. x] by simp

lemma split-vars-into-subst-map-vars-term:
  assumes split: split-vars l = (C, xs)
  and len: length ts = length xs
  and id: ∧ i. i < length xs ⇒ σ (xs ! i) = ts ! i
  shows l · σ =f (map-vars-mtxt vw C, ts)
proof -
  from split-vars-eqf-subst-map-vars-term[of l σ vw, unfolded split]
  have l · σ =f (map-vars-mtxt vw C, map σ xs) by simp
  also have map σ xs = ts
  by (rule nth-equalityI, insert len id, auto)
  finally show ?thesis .
qed

lemma split-vars-into-subst:
  assumes split: split-vars l = (C, xs)
  and len: length ts = length xs
  and id: ∧ i. i < length xs ⇒ σ (xs ! i) = ts ! i
  shows l · σ =f (C, ts)
  using split-vars-into-subst-map-vars-term[OF split len id, of λ x. x] by simp

lemma eqf-funas-term:
  t =f (C, ss) ⇒ funas-term t = funas-mtxt C ∪ ∪ (funas-term ' set ss)
proof (induct C arbitrary: t ss)
  case (MFun f Cs t ss)
  from eqf-MFunE[OF MFun(2)] obtain ts sss where
    t: t = Fun f ts and len: length ts = length Cs length sss = length Cs
    and args: ∧ i. i < length Cs ⇒ ts ! i =f (Cs ! i, sss ! i)
    and ss: ss = concat sss by auto
  let ?lhs = ∪ {funas-term (ts ! i) | i. i < length Cs}
  let ?f1 = λ i. funas-mtxt (Cs ! i)
  let ?f2 = λ i. ∪ (funas-term ' set (sss ! i))
  let ?f = λ i. ?f1 i ∪ ?f2 i
  {

```

```

    fix i
    assume i: i < length Cs
    then have mem: Cs ! i ∈ set Cs by auto
    note MFun(1)[OF mem args[OF i]]
  } note IH = this
  have funas-term t = insert (f,length Cs) ?lhs
    unfolding t using len by (auto simp: set-conv-nth)
  also have ?lhs =  $\bigcup \{?f\ i \mid i. i < \text{length } Cs\}$  using IH by blast
  also have ... =  $\bigcup \{?f1\ i \mid i. i < \text{length } Cs\} \cup \bigcup \{?f2\ i \mid i. i < \text{length } Cs\}$  by
  auto
    also have insert (f,length Cs) ... = (insert (f,length Cs) ( $\bigcup \{?f1\ i \mid i. i < \text{length } Cs\}$ ))  $\cup \bigcup \{?f2\ i \mid i. i < \text{length } Cs\}$  by auto
    also have insert (f,length Cs) ( $\bigcup \{?f1\ i \mid i. i < \text{length } Cs\}$ ) = funas-mctxt
    (MFun f Cs)
      by (auto simp: set-conv-nth)
    also have  $\bigcup \{?f2\ i \mid i. i < \text{length } Cs\} = \bigcup (\text{funas-term } ' \text{ set } ss)$  unfolding ss
    len(2)[symmetric]
      using set-conv-nth[of sss] by auto
    finally show ?case .
next
  case MVar
  from eqfE[OF this]
  show ?case by auto
next
  case MHole
  from eqfE[OF this] show ?case by (cases ss, auto)
qed

lemma eqf-all-ctxt-closed-step:
  assumes ctxt: all-ctxt-closed F R
  and ass:  $t =_f (D,ss) \wedge i. i < \text{length } ts \implies (ss ! i, ts ! i) \in R$   $\text{length } ss = \text{length } ts$ 
  funas-term  $t \subseteq F$ 
   $\bigcup (\text{funas-term } ' \text{ set } ts) \subseteq F$ 
  shows  $(t, \text{fill-holes } D\ ts) \in R \wedge \text{fill-holes } D\ ts =_f (D, ts)$ 
using ass
proof (induct t (D,ss) rule: eq-fill.induct)
  case (eqfI t)
  from eqfI(2) eqfI(4)[unfolded eqfI(2)[symmetric]] eqfI(3,5,6)
  show ?case unfolding eqfI(1)
  proof (induct D ss ts rule: fill-holes-induct2)
    case (MVar v) then show ?case using all-ctxt-closed-sig-refl[OF ctxt] by
    auto
  next
    case (MHole s' t') then show ?case by auto
  next
    case (MFun f Cs ss ts)
    let ?ss = (map ( $\lambda i. \text{fill-holes } (Cs ! i) (\text{partition-holes } ss\ Cs ! i)$ ) [0.. $\text{length } Cs$ ])
    let ?ts = (map ( $\lambda i. \text{fill-holes } (Cs ! i) (\text{partition-holes } ts\ Cs ! i)$ ) [0.. $\text{length } ts$ ])

```

```

Cs])
  note * = all-ctxt-closedD[OF ctxt, of f ?ts ?ss, unfolded length-map length-upt
minus-nat.diff-0]
  show ?case unfolding fill-holes.simps MFun(4) concat-partition-by[OF MFun(1)]
concat-partition-by[OF MFun(2)]
  proof (intro conjI eqfI *)
    fix i assume i: i < length Cs
    then have *: i < length ?ss i < length ?ts by auto
    from *(1) MFun(1,5) have g1: funas-term (fill-holes (Cs ! i) (partition-holes
ss Cs ! i)) ⊆ F
    by (auto simp: subset-eq)
    with *(1) show funas-term (?ss ! i) ⊆ F by auto
    from *(2) MFun(2,6) have g2: (⋃ a∈set (partition-holes ts Cs ! i). funas-term
a) ⊆ F
    unfolding set-concat
    by (auto simp: subset-eq all-set-conv-all-nth[of partition-holes ts Cs])
    with *(2) MFun(1,2,5) show funas-term (?ts ! i) ⊆ F
    by (auto simp: funas-term-fill-holes-iff subset-eq)
    {
      fix j assume j: j < length (partition-holes ts Cs ! i)
      from partition-by-nth-nth[of map num-holes Cs ss i j]
      partition-by-nth-nth[of map num-holes Cs ts i j]
      i j MFun(1,2,4)
      have (partition-holes ss Cs ! i ! j, partition-holes ts Cs ! i ! j) ∈ R by simp
    }
    with i show (?ss ! i, ?ts ! i) ∈ R by (auto intro!: conjunct1[OF MFun(3)][OF
i - g1 g2])
  next
    show (f, length Cs) ∈ F using MFun(5) by auto
  next
    show Fun f ?ts =f (MFun f Cs, ts) using MFun(2) by (intro eq-fill.intros)
auto
  qed simp
  qed
  qed

fun map-mctxt :: ('f ⇒ 'g) ⇒ ('f, 'v) mctxt ⇒ ('g, 'v) mctxt
where
  map-mctxt - (MVar x) = (MVar x) |
  map-mctxt - (MHole) = MHole |
  map-mctxt fg (MFun f Cs) = MFun (fg f) (map (map-mctxt fg) Cs)

fun ground-mctxt :: ('f, 'v) mctxt ⇒ bool
where
  ground-mctxt (MVar -) = False |
  ground-mctxt MHole = True |
  ground-mctxt (MFun f Cs) = Ball (set Cs) ground-mctxt

lemma ground-cap-till-funas [intro]:

```

```

    ground-mctxt (cap-till-funas F t)
  by (induct t) simp-all

lemma ground-eq-fill:  $t =_f (C, ss) \implies \text{ground } t = (\text{ground-mctxt } C \wedge (\forall s \in \text{set } ss. \text{ground } s))$ 
proof (induct C arbitrary: t ss)
  case (MVar x)
  from eqfE[OF this] show ?case by simp
next
  case (MHole t ss)
  from eqfE[OF this] show ?case by (cases ss, auto)
next
  case (MFun f Cs s ss)
  from eqfMFunE[OF MFun(2)] obtain ts sss where  $s: s = \text{Fun } f \text{ ts}$  and  $\text{len: length } ts = \text{length } Cs \text{ length } sss = \text{length } Cs$ 
  and IH:  $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$  and  $ss: ss = \text{concat } sss$  by metis
  {
    fix i
    assume  $i: i < \text{length } Cs$ 
    then have  $Cs ! i \in \text{set } Cs$  by simp
    from MFun(1)[OF this IH[OF i]]
    have  $\text{ground } (ts ! i) = (\text{ground-mctxt } (Cs ! i) \wedge (\forall a \in \text{set } (sss ! i). \text{ground } a))$  .
  } note IH = this
  note conv = set-conv-nth
  have ?case =  $((\forall x \in \text{set } ts. \text{ground } x) = ((\forall x \in \text{set } Cs. \text{ground-mctxt } x) \wedge (\forall a \in \text{set } sss. \forall x \in \text{set } a. \text{ground } x)))$ 
  unfolding s ss by simp
  also have ... unfolding conv[of ts] conv[of Cs] conv[of sss] len using IH by
  auto
  finally show ?case by simp
qed

lemma ground-fill-holes:
  assumes  $nh: \text{num-holes } C = \text{length } ss$ 
  shows  $\text{ground } (\text{fill-holes } C ss) = (\text{ground-mctxt } C \wedge (\forall s \in \text{set } ss. \text{ground } s))$ 
  by (rule ground-eq-fill[OF eqfI[OF refl nh]])

lemma split-vars-ground:  $\text{split-vars } t = (C, xs) \implies \text{ground-mctxt } C$ 
proof (induct t arbitrary: C xs)
  case (Fun f ts C xs)
  from Fun(2)[simplified] obtain Cs where  $C: C = \text{MFun } f \text{ Cs}$  and  $Cs: Cs = \text{map } (fst \circ \text{split-vars}) \text{ ts}$  by auto
  show ?case unfolding C ground-mctxt.simps
  proof
    fix C
    assume  $C \in \text{set } Cs$ 
    from this[unfolded Cs] obtain t where  $t: t \in \text{set } ts$  and  $C: C = \text{fst } (\text{split-vars } t)$ 
    unfolding o-def by auto
  end
end

```

```

    from C obtain xs where split: split-vars t = (C,xs) by (cases split-vars t,
auto)
    show ground-mtxt C
    by (rule Fun(1)[OF t split])
qed
qed auto

lemma split-vars-ground-vars:
  assumes ground-mtxt C and num-holes C = length xs
  shows split-vars (fill-holes C (map Var xs)) = (C, xs)
using assms
proof (induct C arbitrary: xs)
  case (MHole xs)
  then show ?case by (cases xs, auto)
next
  case (MFun f Cs xs)
  have fill-holes (MFun f Cs) (map Var xs) =f (MFun f Cs, map Var xs)
  by (rule eqfI, insert MFun(3), auto)
  from eqf-MFunE[OF this]
  obtain ts xss where fh: fill-holes (MFun f Cs) (map Var xs) = Fun f ts
  and lent: length ts = length Cs
  and lenx: length xss = length Cs
  and args:  $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, xss ! i)$ 
  and id: map Var xs = concat xss by auto
  from arg-cong[OF id, of map the-Var] have id2: xs = concat (map (map the-Var)
xss)
  by (metis map-concat length-map map-nth-eq-conv term.sel(1))
  {
    fix i
    assume i: i < length Cs
    then have mem: Cs ! i  $\in$  set Cs by auto
    with MFun(2) have ground: ground-mtxt (Cs ! i) by auto
    have map Var (map the-Var (xss ! i)) = map id (xss ! i) unfolding map-map
o-def map-eq-conv
  proof
    fix x
    assume x  $\in$  set (xss ! i)
    with lenx i have x  $\in$  set (concat xss) by auto
    from this[unfolded id[symmetric]] show Var (the-Var x) = id x by auto
  qed
  then have idxss: map Var (map the-Var (xss ! i)) = xss ! i by auto
  note rec = eqfE[OF args[OF i]]
  note IH = MFun(1)[OF mem ground, of map the-Var (xss ! i), unfolded rec(2)
idxss rec(1)[symmetric]]
  from IH have split-vars (ts ! i) = (Cs ! i, map the-Var (xss ! i)) by auto
  note this idxss
}
  note IH = this
  have ?case = (map fst (map split-vars ts) = Cs  $\wedge$  concat (map snd (map split-vars

```

```

ts)) = concat (map (map the-Var) xss))
  unfolding fh unfolding id2 by auto
also have ...
  proof (rule conjI[OF nth-equalityI arg-cong[of - - concat, OF nth-equalityI,
rule-format]], unfold length-map lent lenx)
    fix i
    assume i: i < length Cs
    with arg-cong[OF IH(2)[OF this], of map the-Var]
      IH[OF this] show map snd (map split-vars ts) ! i = map (map the-Var) xss
  ! i using lent lenx by auto
  qed (insert IH lent, auto)
  finally show ?case .
qed auto

```

```

lemma ground-map-mtxt[simp]: ground-mtxt (map-mtxt fg C) = ground-mtxt C
  by (induct C, auto)

```

```

lemma num-holes-map-mtxt[simp]: num-holes (map-mtxt fg C) = num-holes C
proof (induct C)
  case (MFun f Cs)
  then show ?case by (induct Cs, auto)
qed auto

```

```

lemma split-vars-map-mtxt:
  assumes split: split-vars t = (map-mtxt fg C, xs)
  shows split-vars (fill-holes C (map Var xs)) = (C, xs)
proof -
  from split-vars-ground[OF split] have ground: ground-mtxt C by simp
  from split-vars-num-holes[of t, unfolded split] have nh: num-holes C = length xs
  by auto
  show ?thesis
    by (rule split-vars-ground-vars[OF ground nh])
qed

```

```

lemma subst-eq-map-decomp:
  assumes t · σ = map-funs-term fg s
  shows ∃ C xs δs. s =f (C, δs) ∧ split-vars t = (map-mtxt fg C, xs) ∧ (∀ i <
length xs.
  σ (xs ! i) = map-funs-term fg (δs ! i))
using assms
proof (induct t arbitrary: s)
  case (Var x s)
  show ?case
    by (intro exI[of - MHole] exI[of - [x]] exI[of - [s]], insert Var, auto)
next
  case (Fun g ts s)
  from Fun(2) obtain f ss where s: s = Fun f ss and g: g = fg f by (cases s,
auto)

```

```

from  $Fun(2)[unfolded\ s]$  have  $id: map\ (\lambda\ t.\ t \cdot \sigma)\ ts = map\ (map-funs-term\ fg)$ 
 $ss$  by auto
from  $arg-cong[OF\ this,\ of\ length]$  have  $len: length\ ts = length\ ss$  by auto
from  $map-nth-conv[OF\ id]$  have  $args: \bigwedge i.\ i < length\ ts \implies ts\ !\ i \cdot \sigma =$ 
 $map-funs-term\ fg\ (ss\ !\ i)$  by auto
let  $?P = \lambda\ C\ xs\ \delta s\ i.\ ss\ !\ i =_f (C,\ \delta s) \wedge$ 
 $split-vars\ (ts\ !\ i) = (map-mtxt\ fg\ C,\ xs) \wedge$ 
 $(\forall i < length\ xs.\ \sigma\ (xs\ !\ i) = map-funs-term\ fg\ (\delta s\ !\ i))$ 
{
  fix  $i$ 
  assume  $i: i < length\ ts$ 
  then have  $mem: ts\ !\ i \in set\ ts$  by auto
  note  $IH = Fun(1)[OF\ this\ args[OF\ i]]$ 
}
then have  $\forall i.\ \exists C\ xs\ \delta s.\ i < length\ ts \longrightarrow ?P\ C\ xs\ \delta s\ i$  by blast
from  $choice[OF\ this]$  obtain  $Cs$  where  $\forall i.\ \exists xs\ \delta s.\ i < length\ ts \longrightarrow ?P\ (Cs$ 
 $i)\ xs\ \delta s\ i$  by blast
from  $choice[OF\ this]$  obtain  $xss$  where  $\forall i.\ \exists \delta s.\ i < length\ ts \longrightarrow ?P\ (Cs\ i)$ 
 $(xss\ i)\ \delta s\ i$  by blast
from  $choice[OF\ this]$  obtain  $\delta ss$  where  $IH: \bigwedge i.\ i < length\ ts \implies ?P\ (Cs\ i)$ 
 $(xss\ i)\ (\delta ss\ i)\ i$  by blast
let  $?n = [0 ..< length\ ts]$ 
let  $?Cs = map\ Cs\ ?n$ 
let  $?C = MFun\ f\ ?Cs$ 
let  $?xs = concat\ (map\ xss\ ?n)$ 
let  $?ds = concat\ (map\ \delta ss\ ?n)$ 
let  $?g = fg\ f$ 
show  $?case\ unfolding\ s\ g$ 
proof ( $rule\ exI[of\ -\ ?C]$ ,  $rule\ exI[of\ -\ ?xs]$ ,  $rule\ exI[of\ -\ ?ds]$ ,  $intro\ conjI$ )
  show  $Fun\ f\ ss =_f\ (?C,\ ?ds)$ 
  by ( $rule\ eqf-MFunI$ ,  $insert\ IH\ len$ , auto)
next
have
 $(split-vars\ (Fun\ ?g\ ts) = (map-mtxt\ fg\ ?C,\ ?xs))$ 
 $= (map\ (fst \circ split-vars)\ ts = map\ (map-mtxt\ fg \circ Cs)\ [0..<length\ ss]$ 
 $\wedge concat\ (map\ (snd \circ split-vars)\ ts) = ?xs)$ 
 $(is\ ?goal = -)$ 
using  $len$  by auto
also have ...
  by ( $rule\ conjI[OF\ nth-map-conv\ arg-cong[of\ -\ -\ concat,\ OF\ nth-equalityI]]$ ,
 $insert\ IH\ len$ , auto)
finally show  $?goal$  .
next
show  $\forall i < length\ ?xs.\ \sigma\ (?xs\ !\ i) = map-funs-term\ fg\ (?ds\ !\ i)$ 
proof ( $rule\ concat-all-nth$ ,  $unfold\ length-map\ length-upt$ )
  fix  $i$ 
  assume  $i < length\ ts - 0$ 
  then have  $i: i < length\ ts$  by auto
  from  $IH[OF\ i]$  have  $split-vars\ (ts\ !\ i) = (map-mtxt\ fg\ (Cs\ i),\ xss\ i)$  by blast

```

```

      from split-vars-map-mctxt[OF this] split-vars-num-holes[of fill-holes (Cs i)
(map Var (xss i))]
      have len: length (xss i) = num-holes (Cs i) by simp
      also have ... = length (δss i) by (rule eqfE(2), insert IH[OF i], auto)
      finally
      show length (map xss [0..\implies
  map-funs-term fg (fill-holes C ss) =f (map-mctxt fg C, map (map-funs-term fg)
ss)
proof (induct C arbitrary: ss)
  case (MHole ss)
  then show ?case by (cases ss, auto)
next
  case MVar then show ?case by auto
next
  case (MFun f Cs ss)
  from MFun(2) have fill-holes (MFun f Cs) ss =f (MFun f Cs, ss) by auto
  from eqf-MFunE[OF this] obtain ts sss where fh: fill-holes (MFun f Cs) ss =
Fun f ts
  and lts: length ts = length Cs
  and lsss: length sss = length Cs
  and args:  $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$ 
  and sss: ss = concat sss by auto
  {
    fix i
    assume i: i < length Cs
    then have mem: Cs ! i  $\in$  set Cs by auto
    from MFun(1)[OF mem] eqfE[OF args[OF i]] have
      map-funs-term fg (ts ! i) =f (map-mctxt fg (Cs ! i), map (map-funs-term fg)
(sss ! i)) by auto
    } note IH = this
  show ?case unfolding fh
    unfolding map-mctxt.simps sss map-concat term.simps
  proof (rule eqf-MFunI, unfold length-map)
    fix i
    assume i: i < length Cs
    have (map (map-funs-term fg) ts ! i =f (map (map-mctxt fg) Cs ! i, map (map
(map-funs-term fg)) sss ! i)) =
      (map-funs-term fg (ts ! i) =f (map-mctxt fg (Cs ! i), map (map-funs-term fg)
(sss ! i))) (is ?goal = -)
    using i lts lsss by auto
    also have ... by (rule IH[OF i])
    finally show ?goal .
  
```

qed (*auto simp: lss ls*)
qed

lemma *eqf-MVarE*:
assumes $s =_f (MVar\ x, ss)$
shows $s = Var\ x\ ss = []$
by (*insert eqfE[OF assms], cases s; cases ss, auto*)**+**

lemma *eqf-imp-subt*:
assumes $s: s =_f (C, ts)$
and $t: t \in set\ ts$
shows $s \supseteq t$
proof –
from t **obtain** $bef\ aft$ **where** $ts: ts = bef @ t \# aft$
by (*metis split-list*)
note $s = eqfE[OF\ s[unfolded\ ts], simplified]$
from *fill-holes-ctxt-main*[*OF s(2)*] **obtain** D **where** *fill-holes C (bef @ t # aft)*
 $= D\langle t \rangle$ **by** *auto*
from *this*[*folded s(1)*] **show** *?thesis* **by** *auto*
qed

lemma *eqf-MFun-imp-strict-subt*:
assumes $s:s =_f (MFun\ f\ cs, ts)$
and $t:t \in set\ ts$
shows $s \supset t$
proof –
from t **obtain** $bef\ aft$ **where** $ts: ts = bef @ t \# aft$
by (*metis split-list*)
from *eqfE*[*OF s[unfolded ts]*] **have** $s: s = fill-holes\ (MFun\ f\ cs)\ (bef @ t \# aft)$
 $num-holes\ (MFun\ f\ cs) = Suc\ (length\ bef + length\ aft)$ **by** *auto*
from *fill-holes-ctxt-main'*[*OF s(2)*] **obtain** D
where $D: fill-holes\ (MFun\ f\ cs)\ (bef @ t \# aft) = D\langle t \rangle$ **and** $D \neq \square$ **by** *blast*
from *this*[*folded s(1)*] **show** *?thesis* **by** *auto*
qed

fun *poss-mctxt* :: $(f, 'v)\ mctxt \Rightarrow pos\ set$
where
 $poss-mctxt\ (MVar\ x) = \{\square\}$ |
 $poss-mctxt\ MHole = \{\}$ |
 $poss-mctxt\ (MFun\ f\ cs) = \{\square\} \cup \bigcup (set\ (map\ (\lambda\ i.\ (\lambda\ p.\ i \# p))\ ' poss-mctxt\ (cs\ !\ i))\ [0 ..< length\ cs]))$

lemma *poss-mctxt-simp* [*simp*]:
 $poss-mctxt\ (MFun\ f\ cs) = \{\square\} \cup \{i \# p \mid i\ p.\ i < length\ cs \wedge p \in poss-mctxt\ (cs\ !\ i)\}$
by *auto*
declare *poss-mctxt.simps(3)*[*simp del*]

```

lemma poss-mctxt-map-vars-mctxt [simp]:
  poss-mctxt (map-vars-mctxt f C) = poss-mctxt C
by (induct C) auto

fun hole-poss :: ('f, 'v) mctxt ⇒ pos set
where
  hole-poss (MVar x) = {} |
  hole-poss MHole = {} |
  hole-poss (MFun f cs) = ⋃ (set (map (λ i. (λ p. i # p) ' hole-poss (cs ! i)) [0
    ..< length cs]))

lemma hole-poss-simp [simp]:
  hole-poss (MFun f cs) = {i # p | i p. i < length cs ∧ p ∈ hole-poss (cs ! i)}
  by auto
declare hole-poss.simps(3)[simp del]

lemma hole-poss-empty-iff-num-holes-0: hole-poss C = {} ⟷ num-holes C = 0
  by (induct C; fastforce simp: set-conv-nth)

lemma mctxt-of-term-fill-holes [simp]:
  fill-holes (mctxt-of-term t) [] = t
proof (induct t)
  case (Fun f ts)
  then have fill-holes (mctxt-of-term (Fun f ts)) [] = Fun f (map (λi. (ts!i))
    [0..length ts])
  unfolding mctxt-of-term.simps partition-holes-fill-holes-conv partition-by-Nil
  map-map by auto
  also have ... = Fun f ts using map-nth by auto
  ultimately show ?case by auto
qed (auto)

lemma hole-pos-not-in-poss-mctxt:
  assumes p ∈ hole-poss C
  shows p ∉ poss-mctxt C
using assms
by (induct C arbitrary: p) auto

lemma hole-pos-in-filled-fun-poss:
  assumes is-Fun t
  shows hole-pos E ∈ fun-poss ((E ·c σ)(t · σ))
using assms
by (induct E) (auto simp: append-Cons-nth-middle)

fun
  subst-apply-mctxt :: ('f, 'v) mctxt ⇒ ('f, 'v, 'w) gsubst ⇒ ('f, 'w) mctxt (infixl
    ·mc 67)
where
  MHole ·mc - = MHole |
  (MVar x) ·mc σ = mctxt-of-term (σ x) |

```

```

(MFun f cs) · mc σ = MFun f [c · mc σ . c ← cs]

lemma subst-apply-mctxt-compose: C · mc σ · mc δ = C · mc σ ∘s δ
proof (induct C)
  case (MVar x)
  define t where t = σ x
  show ?case by (simp add: t-def[symmetric] subst-compose-def, induct t, auto)
qed auto

lemma subst-apply-mctxt-cong: (⋀ x. x ∈ vars-mctxt C ⇒ σ x = τ x) ⇒ C
· mc σ = C · mc τ
  by (induct C, auto)

lemma vars-mctxt-subst: vars-mctxt (C · mc σ) = ⋃ (vars-term ‘ σ ‘ vars-mctxt
C)
  by (induct C, auto)

lemma subst-apply-mctxt-numholes:
  shows num-holes (c · mc σ) = num-holes c
proof (induct c arbitrary: σ)
  case (MFun f cs)
  have num-holes (MFun f cs · mc σ) = sum-list [num-holes (c · mc σ) . c ← cs]
    unfolding subst-apply-mctxt.simps num-holes.simps map-map comp-def by
  auto
  also have ... = sum-list [num-holes c . c ← cs] using MFun(1)
    by (metis (lifting, no-types) map-cong)
  ultimately show ?case by auto
qed (auto)

lemma subst-apply-mctxt-fill-holes:
  assumes nh: num-holes c = length ts
  shows (fill-holes c ts) · σ = fill-holes (c · mc σ) [ti · σ . ti ← ts]
using nh
proof (induct c arbitrary: ts)
  case MHole
  then obtain t where ts: ts = [t]
    unfolding num-holes.simps unfolding One-nat-def using Suc-length-conv
  length-0-conv by metis
  show ?case unfolding ts by simp
next
  case (MVar x)
  then have ts: ts = [] using length-0-conv by auto
  show ?case unfolding ts by auto
next
  case (MFun f cs)
  note IH = MFun(1)
  note nh = MFun(2)[unfolded num-holes.simps]
  let ?c = MFun f cs

```

```

let ?csσ = map (λc. c · mc σ) cs

{
  fix i
  assume i: i < length cs
  have nh-map: ∧ j. j < length cs ⇒ num-holes (cs!j) = num-holes (?csσ ! j)
    using nth-map subst-apply-mctxt-numholes by metis

  have fill-holes (cs ! i) (partition-holes ts cs ! i) · σ =
    fill-holes ((cs ! i) · mc σ) (partition-holes [ti · σ . ti ← ts] cs ! i)
    using IH [OF nth-mem [OF i]] and nh and i by auto
  also have ... = fill-holes (?csσ ! i) (partition-holes [ti · σ . ti ← ts] ?csσ ! i)
    unfolding nth-map[OF i] using partition-holes-map-ctxt[OF - nh-map]
length-map by metis
  ultimately have fill-holes (cs ! i) (partition-holes ts cs ! i) · σ = fill-holes
(?csσ ! i) (partition-holes [ti · σ . ti ← ts] ?csσ ! i)
    by auto
  } note ith = this

  have fill-holes ?c ts · σ = Fun f [fill-holes (?csσ ! i) (partition-holes [ti · σ . ti
← ts] ?csσ ! i) . i ← [0..<length cs]]
  unfolding partition-holes-fill-holes-conv map-map using ith using comp-def by
auto
  also have ... = fill-holes (?c · mc σ) [ti · σ . ti ← ts]
  unfolding subst-apply-mctxt.simps partition-holes-fill-holes-conv length-map ..
  ultimately show ?case by auto
qed

lemma subst-apply-mctxt-sound:
  assumes t =f (c, ts)
  shows t · σ =f (c · mc σ, [ti · σ . ti ← ts])
proof (rule eqfI, insert subst-apply-mctxt-numholes subst-apply-mctxt-fill-holes[OF
eqfE(2)[OF assms]] eqfE[OF assms] eqfE(2)[OF assms, symmetric], auto) qed

fun fill-holes-mctxt :: ('f, 'v) mctxt ⇒ ('f, 'v) mctxt list ⇒ ('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 (λ i. fill-holes-mctxt (cs ! i)
(partition-holes ts cs ! i)) [0 ..< length cs]))

lemma fill-holes-mctxt-Nil [simp]:
  fill-holes-mctxt C [] = C
  by (induct C) (auto intro: nth-equalityI)

lemma map-fill-holes-mctxt-zip [simp]:
  assumes length ts = n
  shows map (λ(x, y). fill-holes-mctxt x y) (zip (map mctxt-of-term ts) (replicate

```

$n \ [])) =$
 $\text{map mctxt-of-term } ts$
using *assms* **by** (*induct ts arbitrary: n*) *auto*

lemma *fill-holes-mctxt-MHole* [*simp*]:
 $\text{length } ts = \text{Suc } 0 \implies \text{fill-holes-mctxt MHole } ts = \text{hd } ts$
by (*cases ts*) *simp-all*

lemma *partition-holes-fill-holes-mctxt-conv*:
 $\text{fill-holes-mctxt (MFun f Cs) } ts =$
 $\text{MFun f [fill-holes-mctxt (Cs ! i) (partition-holes ts Cs ! i). i} \leftarrow [0 \dots \text{length}$
 $\text{Cs}]$
by (*simp add: partition-by-nth take-map*)

lemma *partition-holes-fill-holes-mctxt-conv'*:
 $\text{fill-holes-mctxt (MFun f Cs) } ts =$
 $\text{MFun f (map (case-prod fill-holes-mctxt) (zip Cs (partition-holes ts Cs)))}$
unfolding *zip-nth-conv* [*of Cs partition-holes ts Cs, simplified*]
and *partition-holes-fill-holes-mctxt-conv* **by** *simp*

lemma *fill-holes-mctxt-mctxt-of-ctxt-mctxt-of-term* [*simp*]:
 $\text{fill-holes-mctxt (mctxt-of-ctxt C) [mctxt-of-term t]} = \text{mctxt-of-term (C\langle t \rangle)}$
by (*induct C arbitrary: t*)
(simp-all del: fill-holes-mctxt.simps add: partition-holes-fill-holes-mctxt-conv')

lemma *fill-holes-mctxt-mctxt-of-ctxt-MHole* [*simp*]:
 $\text{fill-holes-mctxt (mctxt-of-ctxt C) [MHole]} = \text{mctxt-of-ctxt C}$
by (*induct C*) (*simp-all del: fill-holes-mctxt.simps add: partition-holes-fill-holes-mctxt-conv'*)

lemma *partition-holes-fill-holes-conv'*:
 $\text{fill-holes (MFun f Cs) } ts =$
 $\text{Fun f (map (case-prod fill-holes) (zip Cs (partition-holes ts Cs)))}$
unfolding *zip-nth-conv* [*of Cs partition-holes ts Cs, simplified*]
and *partition-holes-fill-holes-conv* **by** *simp*

lemma *fill-holes-mctxt-MFun-replicate-length* [*simp*]:
 $\text{fill-holes-mctxt (MFun c (replicate (length Cs) MHole)) Cs} = \text{MFun c Cs}$
unfolding *partition-holes-fill-holes-mctxt-conv'*
by (*induct Cs*) *simp-all*

lemma *fill-holes-MFun-replicate-length* [*simp*]:
 $\text{fill-holes (MFun c (replicate (length ts) MHole)) ts} = \text{Fun c ts}$
unfolding *partition-holes-fill-holes-conv'*
by (*induct ts*) *simp-all*

lemma *funas-mctxt-fill-holes-mctxt* [*simp*]:
assumes $\text{num-holes C} = \text{length Ds}$
shows $\text{funas-mctxt (fill-holes-mctxt C Ds)} = \text{funas-mctxt C} \cup \bigcup (\text{set (map funas-mctxt Ds)})$

```

  (is ?f C Ds = ?g C Ds)
using assms
proof (induct C arbitrary: Ds)
  case MHole
  then show ?case by (cases Ds) simp-all
next
  case (MFun f Cs)
  then have num-holes: sum-list (map num-holes Cs) = length Ds by simp
  let ?ys = partition-holes Ds Cs
  have  $\bigwedge i. i < \text{length } Cs \implies ?f (Cs ! i) (?ys ! i) = ?g (Cs ! i) (?ys ! i)$ 
  using MFun by (metis nth-mem num-holes.simps(3) length-partition-holes-nth)
  then have  $(\bigcup i \in \{0 ..< \text{length } Cs\}. ?f (Cs ! i) (?ys ! i)) =$ 
 $(\bigcup i \in \{0 ..< \text{length } Cs\}. ?g (Cs ! i) (?ys ! i))$  by simp
  then show ?case
  using num-holes
  unfolding partition-holes-fill-holes-mctxt-conv
  by (simp add: UN-Un-distrib UN-upt-len-conv [of -  $\lambda x. \bigcup (set x)$ ] UN-set-partition-by-map)
qed simp

```

```

lemma fill-holes-mctxt-MFun:
  assumes lCs: length Cs = length ts
  and lss: length ss = length ts
  and rec:  $\bigwedge i. i < \text{length } ts \implies \text{num-holes } (Cs ! i) = \text{length } (ss ! i) \wedge$ 
 $\text{fill-holes-mctxt } (Cs ! i) (ss ! i) = ts ! i$ 
  shows fill-holes-mctxt (MFun f Cs) (concat ss) = MFun f ts
  unfolding fill-holes-mctxt.simps mctxt.simps
  by (rule conjI[OF refl], rule fill-holes-arbitrary[OF lCs lss rec])

```

```

lemma num-holes-fill-holes-mctxt:
  assumes num-holes C = length Ds
  shows num-holes (fill-holes-mctxt C Ds) = sum-list (map num-holes Ds)
using assms
proof (induct C arbitrary: Ds)
  case MHole
  then show ?case by (cases Ds) simp-all
next
  case (MFun f Cs)
  then have *: map (num-holes  $\circ (\lambda i. \text{fill-holes-mctxt } (Cs ! i) (\text{partition-holes } Ds$ 
 $Cs ! i))) [0..<\text{length } Cs] =$ 
 $\text{map } (\lambda i. \text{sum-list } (\text{map num-holes } (\text{partition-holes } Ds Cs ! i))) [0 ..< \text{length } Cs]$ 
  and sum-list (map num-holes Cs) = length Ds
  by simp-all
  then show ?case
  using map-upt-len-conv [of  $\lambda x. \text{sum-list } (\text{map num-holes } x) \text{ partition-holes } Ds$ 
Cs]
  unfolding partition-holes-fill-holes-mctxt-conv by (simp add: *)
qed simp

```

```

lemma fill-holes-mtxt-fill-holes:
  assumes len-ds:  $\text{length } ds = \text{num-holes } c$ 
  and nh:  $\text{num-holes } (\text{fill-holes-mtxt } c \ ds) = \text{length } ss$ 
  shows fill-holes ( $\text{fill-holes-mtxt } c \ ds$ )  $ss =$ 
     $\text{fill-holes } c \ [\text{fill-holes } (ds \ ! \ i) \ (\text{partition-holes } ss \ ds \ ! \ i). \ i \leftarrow [0 \ ..< \ \text{num-holes } c]]$ 
using assms(1)[symmetric] assms(2)
proof (induct c ds arbitrary: ss rule: fill-holes-induct)
  case (MFun f Cs ds ss)
    define qs where  $qs = \text{map } (\lambda i. \text{fill-holes-mtxt } (Cs \ ! \ i) \ (\text{partition-holes } ds \ Cs \ ! \ i)) \ [0..<\text{length } Cs]$ 
    then have qs:  $\bigwedge i. i < \text{length } Cs \implies \text{fill-holes-mtxt } (Cs \ ! \ i) \ (\text{partition-holes } ds \ Cs \ ! \ i) = qs \ ! \ i$ 
       $\text{length } qs = \text{length } Cs$  by auto
    define zs where  $zs = \text{map } (\lambda i. \text{fill-holes } (ds \ ! \ i) \ (\text{partition-holes } ss \ ds \ ! \ i)) \ [0..<\text{length } ds]$ 
    {
      fix i assume i:  $i < \text{length } Cs$ 
      from MFun(1) have *:  $\text{map length } (\text{partition-holes } ds \ Cs) = \text{map num-holes } Cs$  by auto
      have **:  $\text{length } ss = \text{sum-list } (\text{map sum-list } (\text{partition-holes } (\text{map num-holes } ds) \ Cs))$ 
      using MFun(1) MFun(3)[symmetric] num-holes-fill-holes-mtxt[of MFun f Cs ds]
      by (auto simp: comp-def map-map-partition-by[symmetric])
      have partition-by (partition-by ss
         $(\text{map } (\lambda i. \text{num-holes } (\text{fill-holes-mtxt } (Cs \ ! \ i) \ (\text{partition-holes } ds \ Cs \ ! \ i))) \ [0..<\text{length } Cs]) \ ! \ i)$ 
         $(\text{partition-holes } (\text{map num-holes } ds) \ Cs \ ! \ i) = \text{partition-holes } (\text{partition-holes } ss \ ds) \ Cs \ ! \ i$ 
        using i MFun(1) MFun(3) partition-by-partition-by[OF **]
        by (auto simp: comp-def num-holes-fill-holes-mtxt
          intro!: arg-cong[of - - λx. partition-by (partition-by ss x ! -) -] nth-equalityI)
        then have  $\text{map } (\lambda j. \text{fill-holes } (\text{partition-holes } ds \ Cs \ ! \ i \ ! \ j) \ (\text{partition-holes } (\text{partition-holes } ss \ qs \ ! \ i) \ (\text{partition-holes } ds \ Cs \ ! \ i \ ! \ j))) \ [0..<\text{num-holes } (Cs \ ! \ i)] = \text{partition-holes } zs \ Cs \ ! \ i$ 
        using MFun(1,3)
        by (auto simp: zs-def qs-def i comp-def partition-by-nth-nth intro: nth-equalityI)
      }
    then show ?case using MFun by (simp add: qs-def [symmetric] qs zs-def [symmetric])
qed auto

```

```

lemma fill-holes-mtxt-sound:
  assumes len-ds:  $\text{length } ds = \text{num-holes } c$ 
  and len-sss:  $\text{length } sss = \text{num-holes } c$ 
  and len-ts:  $\text{length } ts = \text{num-holes } c$ 
  and insts:  $\bigwedge i. i < \text{length } ds \implies ts!i =_f (ds!i, sss!i)$ 
  shows  $\text{fill-holes } c \ ts =_f (\text{fill-holes-mtxt } c \ ds, \text{concat } sss)$ 

```

```

proof (rule eqfI)
  note l-nh-i = eqfE(2)[OF insts]

  from partition-holes-concat-id[OF l-nh-i] len-ds len-sss
  have concat-sss: partition-holes (concat sss) ds = sss by auto

  then show nh: num-holes (fill-holes-mctxt c ds) = length (concat sss)
    unfolding num-holes-fill-holes-mctxt [OF len-ds [symmetric]] length-concat
    by (metis l-nh-i len-ds len-sss nth-map-conv)

  have ts: ts = [fill-holes (ds ! i) (partition-holes (concat sss) ds ! i) . i ←
    [0..proof (rule nth-equalityI)
    show l-fhs: length ts = length ?fhs unfolding length-map
      by (metis diff-zero len-ts length-upt)
    fix i
    assume i: i < length ts
    then have i': i < length [0..by (metis diff-zero len-ts length-upt)
    show ts!i = ?fhs ! i
      unfolding nth-map[OF i']
      using eqfE(1)[OF insts[unfolded len-ds, OF i[unfolded len-ts]]]
      by (metis concat-sss i' len-ds len-sss map-nth nth-map)
  qed
  note ts = this

  show fill-holes c ts = fill-holes (fill-holes-mctxt c ds) (concat sss)
    unfolding fill-holes-mctxt-fill-holes[OF len-ds nh] ts ..
  qed

lemma poss-mctxt-fill-holes-mctxt:
  assumes p ∈ poss-mctxt C
  shows p ∈ poss-mctxt (fill-holes-mctxt C Cs)
using assms
proof (induct p arbitrary: C Cs)
  case (Cons a p C Cs)
    thus ?case by (cases C, auto)
  next
    case (Nil C Cs)
    thus ?case by (cases C, auto)
  qed

fun compose-mctxt :: ('f, 'v) mctxt ⇒ nat ⇒ ('f, 'v) mctxt ⇒ ('f, 'v) mctxt
where
  compose-mctxt C i Ci =
    fill-holes-mctxt C [(if i = j then Ci else MHole). j ← [0 ..< num-holes C]]

lemma funas-mctxt-compose-mctxt [simp]:
  assumes i < num-holes C

```

```

  shows funas-mctxt (compose-mctxt C i D) = funas-mctxt C ∪ funas-mctxt D
proof -
  let ?Ds = [(if i = j then D else MHole). j ← [0 ..< num-holes C]]
  have num-holes C = length ?Ds by simp
  then show ?thesis using assms by (auto split: if-splits)
qed

lemma compose-mctxt-sound:
  assumes s: s =f (C, bef @ si # aft)
  and si: si =f (Ci, ts)
  and i: i = length bef
  shows s =f (compose-mctxt C i Ci, bef @ ts @ aft)
proof -
  let ?Cs = [ if i = j then Ci else MHole . j ← [0..<num-holes C]]
  let ?ts = bef @ si # aft
  let ?sss = [ [b]. b ← bef ] @ (ts # [ [a]. a ← aft])

  have
    l-Cs : length ?Cs = num-holes C and
    l-ts : length ?ts = num-holes C and
    l-sss : length ?sss = num-holes C
    unfolding length-append length-map list.size(4) using eqfE(2)[OF s] by auto

  have i-le-nh: i < num-holes C unfolding i eqfE(2)[OF s] length-append by
    (auto iff: trans-less-add1)
  have concat-sss: concat ?sss = bef @ ts @ aft by auto

  {
    fix j
    assume j: j < i
    then have j': j < length [0..<num-holes C] using i-le-nh length-upt by auto
    have ?sss!j = [bef!j] by (metis append-Cons-nth-left i j length-map nth-map)
    moreover have ?ts!j = bef!j by (metis append-Cons-nth-left i j)
    moreover from nth-map[OF j'] j' j have ?Cs!j = MHole by force
    ultimately have ?ts!j =f (?Cs!j, ?sss!j) using eqfI by auto
  } note j-le-i = this

  from i-le-nh have ?Cs!i = Ci by auto
  moreover from i-le-nh have ?sss!i = ts by (metis i length-map nth-append-length)
  moreover have ?ts!i = si using nth-append-length i by auto
  ultimately have j-eq-i: ?ts!i =f (?Cs!i, ?sss!i) using si by auto

  {
    fix j
    assume j: j > i and j': j < num-holes C
    then have j'': j < length [0..<num-holes C] by auto

    have j''': (j - i) - 1 < length aft using
      j'[unfolded eqfE(2)[OF s] length-append[of bef] list.size(4)] j

```

```

    unfolding i by auto

  from nth-append[of [ [b]. b ← bef ] - j, unfolded length-map[of - bef] i[symmetric]]

  have ?sss!j = (ts # [ [a]. a ← aft]) ! (j - i) using j by auto
  moreover have ... = [ [a]. a ← aft] ! ((j - i) - 1) using nth-Cons-pos j by
simp
  moreover have ... = [aft ! ((j - i) - 1)] using j''' length-map nth-map by
auto
  ultimately have sssj: ?sss!j = [aft ! ((j - i) - 1)] by auto

  have Csj: ?Cs!j = MHole using nth-map[OF j''] j'' j by force

  have ?ts!j = (si # aft) ! (j - i) unfolding nth-append[of bef] i[symmetric]
using j by simp
  moreover have ... = aft ! ((j - i) - 1) by (metis j neq0-conv nth-Cons'
zero-less-diff)
  ultimately have ?ts!j = aft ! ((j - i) - 1) by auto
  then have ?ts!j =f (?Cs!j, ?sss!j) using sssj Csj by auto
} note j-gr-i = this

  from j-le-i j-eq-i j-gr-i have  $\bigwedge j. j < \text{length } ?Cs \implies ?ts!j =_f (?Cs!j, ?sss!j)$ 
  using l-Cs linorder-neqE-nat by metis

  from fill-holes-mctxt-sound[OF l-Cs l-sss l-ts this, unfolded concat-sss, folded
compose-mctxt.simps]
  show ?thesis unfolding eqfE(1)[OF s] by simp
qed

fun mctxt-fill-partially-mctxts :: ('f, 'v) term list  $\Rightarrow$  ('f, 'v) term list  $\Rightarrow$  ('f, 'v)
mctxt list
where
  mctxt-fill-partially-mctxts [] ts = map mctxt-of-term ts |
  mctxt-fill-partially-mctxts (s # ss) (t # ts) =
    (if s = t then (MHole # mctxt-fill-partially-mctxts ss ts)
     else (mctxt-of-term t # 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

```

```

using assms by (induct rule: subseq-induct2, auto)

lemma mctxt-fill-partially-fills-length [simp]:
  assumes subseq ss ts
  shows length (mctxt-fill-partially-fills ss ts) = length ts
  using assms by (induct rule: subseq-induct2, auto)

lemma mctxt-fill-partially-numholes:
  assumes subseq ss ts
  shows sum-list [num-holes ci . ci ← mctxt-fill-partially-mctxts ss ts] = length ss
proof (induct ss ts rule: subseq-induct2, goal-cases)
  case (3 s ss ts)
  have ls-one:  $\bigwedge as. \text{sum-list } (1 \# as) = \text{sum-list } as + \text{Suc } 0$ 
    by (metis One-nat-def Suc-eq-plus1 Suc-eq-plus1-left sum-list-simps(2))
  from 3 show ?case
    unfolding mctxt-fill-partially-mctxts.simps list.size
    by (metis (full-types) One-nat-def Suc-eq-plus1 ls-one list.map(2) num-holes.simps(2))
next
  case (4 s ss t ts)
  have ls-zero:  $\bigwedge as. \text{sum-list } (0 \# as) = \text{sum-list } as$  by (metis sum-list-simps(2)
monoid-add-class.add.left-neutral)
  have else:
    mctxt-fill-partially-mctxts (s # ss) (t # ts) = mctxt-of-term t # mctxt-fill-partially-mctxts
    (s # ss) ts
    using 4(1) by auto
  show ?case
    unfolding else list.map(2) num-holes-mctxt-of-term ls-zero 4(5) ..
qed (auto iff: assms)

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 (rule eqfI, goal-cases)
  let ?zipped = zip (mctxt-fill-partially-mctxts ss ts) (mctxt-fill-partially-fills ss ts)

  have l: length ?zipped = length ts
  unfolding length-zip mctxt-fill-partially-mctxts-length[OF sl] mctxt-fill-partially-fills-length[OF sl] by auto

  have fh: ts = map ( $\lambda (ci, tsi). \text{fill-holes } ci \text{ } tsi$ ) ?zipped
  proof (induct ss ts rule: subseq-induct2, goal-cases)
    case (2 ts) then show ?case
      by (induct ts, insert mctxt-of-term-fill-holes, auto)
  qed (insert sl, auto)

  have nh: list-all ( $\lambda (ci, tsi). \text{num-holes } ci = \text{length } tsi$ ) ?zipped
  proof (induct ss ts rule: subseq-induct2, goal-cases)
    case (2 ts) then show ?case

```

```

    by (induct ts, insert num-holes-mtxt-of-term, auto)
qed (insert sl, auto)

{
  fix i
  assume i < length ts
  then have
    i1: i < length (mtxt-fill-partially-mtxts ss ts) and
    i2: i < length (mtxt-fill-partially-fills ss ts)
    unfolding mtxt-fill-partially-mtxts-length[OF sl] mtxt-fill-partially-fills-length[OF
sl] by auto
  } note i = this

  case 1
  then show ?case using fh nth-zip[OF i(1) i(2)]
    by (metis (lifting, no-types) 1 list-update-id list-update-same-conv map-update
split-conv)
  case 2 then show ?case using nh[unfolded list-all-length] nth-zip[OF i(1) i(2)]
    by (auto simp: i(1) i(2))
qed

lemma mtxt-fill-partially:
  assumes ss: subseq ss ts
  and t: t =f (c, ts)
  shows ∃ d. t =f (d, ss)
proof -
  let ?ds = mtxt-fill-partially-mtxts ss ts
  let ?sss = mtxt-fill-partially-fills ss ts

  have fill-holes c ts =f (fill-holes-mtxt c ?ds, concat ?sss)
  using
    fill-holes-mtxt-sound eqfE(2)[OF t, symmetric] mtxt-fill-partially-sound[OF
ss]
    mtxt-fill-partially-mtxts-length[OF ss] mtxt-fill-partially-fills-length[OF ss]
  by metis
  also have concat ?sss = ss by (induct ss ts rule: subseq-induct2, insert ss, auto)
  ultimately show ?thesis by (metis eqfE(1) t)
qed

lemma fill-holes-mtxt-map-mtxt-of-term-conv [simp]:
  assumes num-holes C = length ts
  shows fill-holes-mtxt C (map mtxt-of-term ts) = mtxt-of-term (fill-holes C ts)
using assms
by (induct C ts rule: fill-holes-induct) (auto)

lemma fill-holes-mtxt-of-ctxt [simp]:
  fill-holes (mtxt-of-ctxt C) [t] = C⟨t⟩
proof -
  have C⟨t⟩ =f (mtxt-of-ctxt C, [t]) by (metis mtxt-of-ctxt)

```

from *eqfE* [*OF this*] **show** *?thesis* **by** *simp*
qed

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 *compose-cap-till-funas* $F \equiv$ *compose-cap-till* (*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 *fill-holes* (*compose-cap-till* $P\ s\ i\ C$) $ts =$
fill-holes (*cap-till* $P\ s$) (*take* i (*uncap-till* $P\ s$)) @ *fill-holes* $C\ ts\ \#$ *drop* (*Suc* i)
(*uncap-till* $P\ s$)
(is $- = \text{fill-holes} - ?ss$)

proof $-$

have *fill-holes* (*cap-till* $P\ s$) $?ss =_f$
(*fill-holes-mctxt* (*cap-till* $P\ s$) (*map mctxt-of-term* (*take* i (*uncap-till* $P\ s$)) @
 $C\ \#$ *map mctxt-of-term* (*drop* (*Suc* i) (*uncap-till* $P\ s$))),
concat (*map* ($\lambda\cdot. []$) (*take* i (*uncap-till* $P\ s$)) @ $ts\ \#$
map ($\lambda\cdot. []$) (*drop* (*Suc* i) (*uncap-till* $P\ s$))))
(is $- =_f$ (*fill-holes-mctxt* $- ?ts$, *concat* $?us$))

proof (*rule fill-holes-mctxt-sound*)

show $\text{length } ?ss = \text{num-holes } (\text{cap-till } P\ s)$
using *assms* **by** *simp*

next

show $\text{length } ?ts = \text{num-holes } (\text{cap-till } P\ s)$
using *assms* **by** *simp*

next

show $\text{length } ?us = \text{num-holes } (\text{cap-till } P\ s)$
using *assms* **by** *simp*

next

fix j
assume $j < \text{length } ?ts$
with *assms* **have** $j: j < \text{length } (\text{uncap-till } P\ s)$ **by** *simp*
show $?ss ! j =_f (?ts ! j, ?us ! j)$
using *assms* **and** j **by** (*cases* $j = i$) (*auto simp: nth-append*)

qed

note $* = \text{eqfE}(1)$ [*OF this*]

show *?thesis* **by** (*simp add: compose-cap-till-def* $*$)

qed

lemma *in-uncap-till-funas*:

assumes *root*: $\text{root } u = \text{Some } fn\ fn \in F$

and $t = C\langle u \rangle$

shows $\exists i < \text{length } (\text{uncap-till-funas } F\ t). \exists D. \text{uncap-till-funas } F\ t ! i = D\langle u \rangle \wedge$
mctxt-of-ctxt $C = \text{compose-cap-till-funas } F\ t\ i$ (*mctxt-of-ctxt* D)

using $\langle t = C\langle u \rangle \rangle$

```

proof (induct t arbitrary: C)
  case (Var x)
  then show ?case using root by (cases C) (auto simp: wf-trs-def)
next
  case (Fun f ts)
  define t where [simp]: t = Fun f ts
  show ?case
  proof (cases (f, length ts) ∈ F)
    case True
    then show ?thesis using Fun.premis by (auto simp: compose-cap-till-def)
  next
    case False
    show ?thesis
  proof (cases C)
    case Hole
    then show ?thesis using Fun.premis and False and root by auto
  next
    case (More - ss1 D -)
    moreover define j where j = length ss1
    ultimately have j: j < length ts ts ! j = D⟨u⟩
    and C: C = More f (take j ts) D (drop (Suc j) ts)
    using Fun.premis by (auto)
    then have D⟨u⟩ ∈ set ts by (auto simp: in-set-conv-nth)
    then obtain i and E
    where i: i < length (uncap-till-funas F (D⟨u⟩)) uncap-till-funas F (D⟨u⟩) !
    i = E⟨u⟩
    and D: mctxt-of-ctxt D = compose-cap-till-funas F (D⟨u⟩) i (mctxt-of-ctxt
    E)
    using Fun by blast
    obtain k where k: take k (uncap-till-funas F t) =
    concat (map (uncap-till-funas F) (take j ts)) @ take i (uncap-till-funas F
    (D⟨u⟩))
    k < length (uncap-till-funas F t) uncap-till-funas F t ! k = E⟨u⟩
    drop (Suc k) ((uncap-till-funas F) t) = drop (Suc i) ((uncap-till-funas F)
    (D⟨u⟩)) @ concat (map (uncap-till-funas F) (drop (Suc j) ts))
    using False and i and j and take-nth-drop-concat [of j map (uncap-till-funas
    F) ts (uncap-till-funas F) (D⟨u⟩) i E⟨u⟩]
    by (auto simp: take-map drop-map)
    moreover have mctxt-of-ctxt C = compose-cap-till-funas F t k (mctxt-of-ctxt
    E)
  proof -
  have *: compose-cap-till-funas F t k (mctxt-of-ctxt E) =
  fill-holes-mctxt (MFun f (map (cap-till-funas F) ts)) (concat (
  map (map mctxt-of-term ∘ uncap-till-funas F) (take j ts) @
  (map mctxt-of-term (take i (uncap-till-funas F D⟨u⟩)) @
  mctxt-of-ctxt E #
  map mctxt-of-term (drop (Suc i) (uncap-till-funas F D⟨u⟩))) #
  map (map mctxt-of-term ∘ uncap-till-funas F) (drop (Suc j) ts)))
  (is - = fill-holes-mctxt - (concat ?ss))

```

```

    using False and k
    by (simp del: fill-holes-mctxt.simps add: compose-cap-till-def map-concat)
    also have ... = MFun f (map mctxt-of-term (take j ts) @ mctxt-of-ctxt D
#
    map mctxt-of-term (drop (Suc j) ts)) (is - = MFun f ?ts)
  proof (rule fill-holes-mctxt-MFun)
    show length (map (cap-till-funas F) ts) = length ?ts using j by simp
  next
    show length ?ss = length ?ts by simp
  next
    fix n
    assume n < length ?ts
    then have n: n < length ts using j by simp
    show num-holes (map (cap-till-funas F) ts ! n) = length (?ss ! n) ∧
      fill-holes-mctxt (map (cap-till-funas F) ts ! n) (?ss ! n) = ?ts ! n
    proof (cases n = j)
      case False
      then have *: ?ss ! n = map mctxt-of-term (uncap-till-funas F (ts ! n))
        ?ts ! n = mctxt-of-term (ts ! n)
        using n and j by (auto simp: nth-append min-def)
      have num-holes (map (cap-till-funas F) ts ! n) = length (?ss ! n)
        using n by (simp add: *)
      moreover have fill-holes-mctxt (map (cap-till-funas F) ts ! n) (?ss ! n)
= ?ts ! n
        using n by (auto simp: *)
      ultimately show ?thesis by blast
    next
      case True
      then have *: ?ss ! n =
        map mctxt-of-term (take i (uncap-till-funas F D⟨u⟩)) @ mctxt-of-ctxt E
#
        map mctxt-of-term (drop (Suc i) (uncap-till-funas F D⟨u⟩))
        ?ts ! n = mctxt-of-ctxt D
        using n and j by (auto simp: nth-append)
      have fill-holes-mctxt (map (cap-till-funas F) ts ! n) (?ss ! n) = ?ts ! n
        unfolding * by (simp add: D compose-cap-till-def True j)
      moreover have num-holes (map (cap-till-funas F) ts ! n) = length (?ss
! n)
        unfolding * using i by (simp add: j True)
      ultimately show ?thesis by blast
    qed
  qed
  finally show ?thesis by (simp add: C)
  qed
  ultimately show ?thesis unfolding t-def by blast
  qed
  qed
  qed

```

```

lemma uncap-till-funas-fill-holes-cancel [simp]:
  assumes num-holes  $C = \text{length } ts$  and ground-mctxt  $C$ 
    and funas-mctxt  $C \subseteq - F$ 
  shows uncap-till-funas  $F$  (fill-holes  $C$   $ts$ ) = concat (map (uncap-till-funas  $F$ )  $ts$ )
using assms
proof (induct  $C$  arbitrary: ts)
  case MHole
  then show ?case by (cases  $ts$ ) simp-all
next
  case (MFun  $f$   $Cs$ )
  let  $?ts = \text{partition-holes } ts \ Cs$ 
  let  $?us = \text{partition-holes } (\text{map } (\text{uncap-till-funas } F) \ ts) \ Cs$ 
  have *: fill-holes (MFun  $f$   $Cs$ )  $ts =$ 
    Fun  $f$  (map ( $\lambda i. \text{fill-holes } (Cs \ ! \ i) \ (?ts \ ! \ i)$ )  $[0 \ ..< \text{length } Cs]$ )
  unfolding partition-holes-fill-holes-conv ..
  have  $\forall i < \text{length } Cs. \text{uncap-till-funas } F \ (\text{fill-holes } (Cs \ ! \ i) \ (?ts \ ! \ i)) = \text{concat}$ 
    ( $?us \ ! \ i$ )
  proof (intro allI impI)
    fix  $i$ 
    assume  $i < \text{length } Cs$ 
    then have  $Cs \ ! \ i \in \text{set } Cs$  by simp
    from MFun.hyps [OF this, of ?ts ! i] and MFun.prems and  $\langle i < \text{length } Cs \rangle$ 
      show uncap-till-funas  $F$  (fill-holes  $(Cs \ ! \ i) \ (?ts \ ! \ i)$ ) = concat ( $?us \ ! \ i$ )
      by (auto iff: UN-subset-iff)
  qed
  then have **: map (uncap-till-funas  $F \circ (\lambda i. \text{fill-holes } (Cs \ ! \ i) \ (?ts \ ! \ i))$ )
     $[0 \ ..< \text{length } Cs] =$ 
    map (concat  $\circ (\lambda i. (?us \ ! \ i))$ )  $[0 \ ..< \text{length } Cs]$  by simp
  have ***: sum-list (map num-holes  $Cs$ ) = length (map (uncap-till-funas  $F$ )  $ts$ )
    using MFun.prems by simp
  show ?case
    using MFun.prems
    apply (simp add: * ** del: fill-holes.simps)
    by (auto simp: o-def map-upt-len-same-len-conv [OF length-partition-holes])
qed simp

```

```

lemma uncap-till-funas-fill-holes-cap-till-funas [simp]:
  assumes num-holes (cap-till-funas  $F$   $s$ ) = length  $ts$ 
  shows uncap-till-funas  $F$  (fill-holes (cap-till-funas  $F$   $s$ )  $ts$ ) =
    concat (map (uncap-till-funas  $F$ )  $ts$ )
  by (rule uncap-till-funas-fill-holes-cancel [OF assms ground-cap-till-funas, of F])
auto

```

```

lemma Ball-atLeast0LessThan-partition-holes-conv [simp]:
   $(\forall i \in \{0 \ ..< \text{length } Cs\}. \forall x \in \text{set } (\text{partition-holes } xs \ Cs \ ! \ i). P \ x) =$ 
   $(\forall x \in \bigcup (\text{set } (\text{map } \text{set } (\text{partition-holes } xs \ Cs))) . P \ x)$ 
  using Ball-atLeast0LessThan-partition-by-conv [of map num-holes Cs xs] by simp

```

```

lemma ground-fill-holes-mctxt [simp]:

```

$\text{num-holes } C = \text{length } Ds \implies$
 $\text{ground-mctxt } (\text{fill-holes-mctxt } C \ Ds) \longleftrightarrow \text{ground-mctxt } C \wedge (\forall D \in \text{set } Ds.$
 $\text{ground-mctxt } D)$
proof (*induct* C *arbitrary:* Ds)
 case $M\text{Hole}$
 then show $?case$ **by** (*cases* Ds) *simp-all*
next
 case ($M\text{Fun } f \ Cs$)
 then have $*$: $(\forall i \in \{0..<\text{length } Cs\}.$
 $\text{ground-mctxt } (\text{fill-holes-mctxt } (Cs \ ! \ i) \ (\text{partition-holes } Ds \ Cs \ ! \ i))) =$
 $(\forall i \in \{0..<\text{length } Cs\}.$
 $\text{ground-mctxt } (Cs \ ! \ i) \wedge (\forall a \in \text{set } (\text{partition-holes } Ds \ Cs \ ! \ i). \text{ground-mctxt } a))$
 and $**$: $\text{sum-list } (\text{map } \text{num-holes } Cs) = \text{length } Ds$
 by *simp-all*
 show $?case$
 unfolding *partition-holes-fill-holes-mctxt-conv*
 by (*simp add:* $*$ *ball-conj-distrib* *Ball-set-partition-by* [*OF* $**$])
qed *simp*

lemma *concat-map-uncap-till-fun-as-map-subst-apply-uncap-till-fun-as* [*simp*]:
 $\text{concat } (\text{map } (\text{uncap-till-fun-as } F) \ (\text{map } (\lambda s. s \cdot \sigma) \ (\text{uncap-till-fun-as } F \ t))) =$
 $\text{uncap-till-fun-as } F \ (t \cdot \sigma)$
proof (*induct* t)
 case ($\text{Fun } f \ ts$)
 then have $*$: $\text{map } (\text{uncap-till-fun-as } F \circ (\lambda t. t \cdot \sigma)) \ ts =$
 $\text{map } \text{concat } (\text{map } (\text{map } (\text{uncap-till-fun-as } F) \circ \text{map } (\lambda s. s \cdot \sigma) \circ \text{uncap-till-fun-as}$
 $F) \ ts) \text{ by } \text{simp}$
 show $?case$
 by (*simp add:* $*$ *map-concat* *concat-map-concat* [*symmetric*])
qed *simp*

lemma *concat-uncap-till-subst-conv*:
 $\text{concat } (\text{map } (\lambda i. \text{uncap-till-fun-as } F \ ((\text{uncap-till-fun-as } F \ t \ ! \ i) \cdot \sigma)) \ [0 \ ..< \text{length}$
 $(\text{uncap-till-fun-as } F \ t)]) =$
 $\text{uncap-till-fun-as } F \ (t \cdot \sigma)$
proof –
 have $\text{concat } (\text{map } (\text{uncap-till-fun-as } F) \ (\text{map } (\lambda i.$
 $(\text{uncap-till-fun-as } F \ t \ ! \ i) \cdot \sigma) \ [0 \ ..< \text{length } (\text{uncap-till-fun-as } F \ t)])) = \text{un-}$
 $\text{cap-till-fun-as } F \ (t \cdot \sigma)$
 unfolding *map-upt-len-conv* [*of* $\lambda s. s \cdot \sigma$ *uncap-till-fun-as* $F \ t$]
 unfolding *concat-map-uncap-till-fun-as-map-subst-apply-uncap-till-fun-as* ..
 then show $?thesis$ **by** (*simp add:* *o-def*)
qed

lemma *the-root-uncap-till-fun-as*:
 $\text{is-Fun } t \implies \text{the } (\text{root } t) \in F \implies \text{uncap-till-fun-as } F \ t = [t]$
 by (*cases* t) *simp-all*

lemma *fun-as-cap-till-subset*:

```

funas-mctxt (cap-till P t)  $\subseteq$  funas-term t
by (induct t) auto

lemma funas-uncap-till-subset:
   $s \in \text{set } (\text{uncap-till } P \ t) \implies \text{funas-term } s \subseteq \text{funas-term } t$ 
proof (induct t arbitrary: s)
  case (Fun f ts)
  then show ?case by (cases P (Fun f ts)) auto
qed simp

lemma ground-mctxt-subst-apply-context [simp]:
  ground-mctxt C  $\implies C \cdot \text{mc } \sigma = C$ 
by (induct C) (simp-all add: map-idI)

lemma vars-term-fill-holes [simp]:
   $\text{num-holes } C = \text{length } ts \implies \text{ground-mctxt } C \implies$ 
   $\text{vars-term } (\text{fill-holes } C \ ts) = \bigcup (\text{vars-term } \text{' set } ts)$ 
proof (induct C arbitrary: ts)
  case MHole
  then show ?case by (cases ts) simp-all
next
  case (MFun f Cs)
  then have *:  $\text{length } (\text{partition-holes } ts \ Cs) = \text{length } Cs$  by simp
  let ?f =  $\lambda x. \bigcup y \in \text{set } x. \text{vars-term } y$ 
  show ?case
    using MFun
    unfolding partition-holes-fill-holes-conv
    by (simp add: UN-upt-len-conv [OF *, of ?f] UN-set-partition-by)
qed simp

```

7.3 Semilattice Structures

```

instantiation mctxt :: (type, type) inf
begin

fun inf-mctxt :: ('a, 'b) mctxt  $\Rightarrow$  ('a, 'b) mctxt  $\Rightarrow$  ('a, 'b) mctxt
where
  MHole  $\sqcap D = \text{MHole}$  |
  C  $\sqcap \text{MHole} = \text{MHole}$  |
  MVar x  $\sqcap \text{MVar } y = (\text{if } x = y \text{ then } \text{MVar } x \text{ else } \text{MHole})$  |
  MFun f Cs  $\sqcap \text{MFun } g \ Ds =$ 
    ( $\text{if } f = g \wedge \text{length } Cs = \text{length } Ds \text{ then } \text{MFun } f \ (\text{map } (\text{case-prod } (\sqcap)) \ (\text{zip } Cs \ Ds))$ 
     $\text{else } \text{MHole}$ ) |
  C  $\sqcap D = \text{MHole}$ 

instance ..

end

```

```

lemma inf-mtxt-idem [simp]:
  fixes  $C :: ('f, 'v) \text{ mtxt}$ 
  shows  $C \sqcap C = C$ 
  by (induct  $C$ ) (auto simp: zip-same-conv-map intro: map-idI)

lemma inf-mtxt-MHole2 [simp]:
   $C \sqcap \text{MHole} = \text{MHole}$ 
  by (induct  $C$ ) simp-all

lemma inf-mtxt-comm [ac-simps]:
   $(C :: ('f, 'v) \text{ mtxt}) \sqcap D = D \sqcap C$ 
  by (induct  $C$   $D$  rule: inf-mtxt.induct) (fastforce simp: in-set-conv-nth intro!: nth-equalityI) +

lemma inf-mtxt-assoc [ac-simps]:
  fixes  $C :: ('f, 'v) \text{ mtxt}$ 
  shows  $C \sqcap D \sqcap E = C \sqcap (D \sqcap E)$ 
proof (induction  $C$   $D$  arbitrary: E rule: inf-mtxt.induct)
  case (1  $D$   $E$ )
  then show ?case by (cases  $E$ , auto)
next
  case (2-1  $v$   $E$ )
  then show ?case by (cases  $E$ , auto)
next
  case (2-2  $v$   $va$   $E$ )
  then show ?case by (cases  $E$ , auto)
next
  case (3  $x$   $y$   $E$ )
  then show ?case by (cases  $E$ , auto)
next
  case (4  $f$   $Cs$   $g$   $Ds$   $E$ )
  then show ?case
    by (cases  $E$ ; fastforce simp: in-set-conv-nth intro!: nth-equalityI)
next
  case (5-1  $v$   $va$   $vb$   $E$ )
  then show ?case by (cases  $E$ , auto)
next
  case (5-2  $v$   $va$   $vb$   $E$ )
  then show ?case by (cases  $E$ , auto)
qed

instantiation mtxt :: (type, type) order
begin

definition  $(C :: ('a, 'b) \text{ mtxt}) \leq D \iff C \sqcap D = C$ 
definition  $(C :: ('a, 'b) \text{ mtxt}) < D \iff C \leq D \wedge \neg D \leq C$ 

instance

```

by (standard, simp-all add: less-eq-mctxt-def less-mctxt-def ac-simps, metis inf-mctxt-assoc)

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 \Rightarrow ($\bigwedge i. i < \text{length } cs \Rightarrow \text{less-eq-mctxt}' (cs ! i) (ds ! i)$)
 \Rightarrow less-eq-mctxt' (MFun f cs) (MFun f ds)

lemma less-eq-mctxt-prime: $C \leq D \longleftrightarrow \text{less-eq-mctxt}' C D$

proof

assume less-eq-mctxt' C D **then show** $C \leq D$

by (induct C D rule: less-eq-mctxt'.induct) (auto simp: less-eq-mctxt-def intro: nth-equalityI)

next

assume $C \leq D$ **then show** less-eq-mctxt' C D **unfolding** less-eq-mctxt-def

by (induct C D rule: inf-mctxt.induct)

(auto split: if-splits simp: set-zip intro!: less-eq-mctxt'.intros nth-equalityI elim!: nth-equalityE, metis)

qed

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:

$C = \text{MHole} \Rightarrow C \leq \text{MHole}$

$C = \text{MHole} \vee C = \text{MVar } v \Rightarrow C \leq \text{MVar } v$

$C = \text{MHole} \vee C = \text{MFun } f \text{ cs} \wedge \text{length } cs = \text{length } ds \wedge (\forall i. i < \text{length } cs \rightarrow cs ! i \leq ds ! i) \Rightarrow C \leq \text{MFun } f \text{ ds}$

unfolding less-eq-mctxt-prime **by** (cases C) (auto intro: less-eq-mctxt'.intros)

lemma less-eq-mctxt-MHoleE2:

assumes $C \leq \text{MHole}$

obtains (MHole) $C = \text{MHole}$

using assms **unfolding** less-eq-mctxt-prime **by** (cases C, auto)

lemma less-eq-mctxt-MVarE2:

assumes $C \leq \text{MVar } v$

obtains (MHole) $C = \text{MHole}$ | (MVar) $C = \text{MVar } v$

using assms **unfolding** less-eq-mctxt-prime **by** (cases C) auto

lemma less-eq-mctxt-MFunE2:

assumes $C \leq \text{MFun } f \text{ ds}$

obtains (MHole) $C = \text{MHole}$

| (MFun) cs **where** $C = \text{MFun } f \text{ cs}$ length cs = length ds $\wedge \bigwedge i. i < \text{length } cs \Rightarrow cs ! i \leq ds ! i$

using assms **unfolding** less-eq-mctxt-prime **by** (cases C) auto

lemmas *less-eq-mtxtE2* = *less-eq-mtxt-MHoleE2* *less-eq-mtxt-MVarE2* *less-eq-mtxt-MFunE2*

lemma *less-eq-mtxtI1*:

$MHole \leq D$

$D = MVar\ v \implies MVar\ v \leq D$

$D = MFun\ f\ ds \implies length\ cs = length\ ds \implies (\bigwedge i. i < length\ cs \implies cs\ !\ i \leq ds\ !\ i) \implies MFun\ f\ cs \leq D$

by (*cases* *D*) (*auto intro: less-eq-mtxtI2*)

lemma *less-eq-mtxt-MVarE1*:

assumes $MVar\ v \leq D$

obtains $(MVar)\ D = MVar\ v$

using *assms* **by** (*cases* *D*) (*auto elim: less-eq-mtxtE2*)

lemma *less-eq-mtxt-MFunE1*:

assumes $MFun\ f\ cs \leq D$

obtains $(MFun)\ ds$ **where** $D = MFun\ f\ ds\ length\ cs = length\ ds \bigwedge i. i < length\ cs \implies cs\ !\ i \leq ds\ !\ i$

using *assms* **by** (*cases* *D*) (*auto elim: less-eq-mtxtE2*)

lemmas *less-eq-mtxtE1* = *less-eq-mtxt-MVarE1* *less-eq-mtxt-MFunE1*

instance *mtxt* :: (*type*, *type*) *semilattice-inf*

apply (*intro-classes*)

by (*auto simp: less-eq-mtxt-def inf-mtxt-assoc [symmetric]*)

(*metis inf-mtxt-comm inf-mtxt-assoc inf-mtxt-idem*)+

fun *inf-mtxt-args* :: (*f*, *v*) *mtxt* \Rightarrow (*f*, *v*) *mtxt* \Rightarrow (*f*, *v*) *mtxt* *list*

where

inf-mtxt-args *MHole* *D* = [*MHole*] |

inf-mtxt-args *C* *MHole* = [*C*] |

inf-mtxt-args (*MVar* *x*) (*MVar* *y*) = (if *x* = *y* then [] else [*MVar* *x*]) |

inf-mtxt-args (*MFun* *f* *Cs*) (*MFun* *g* *Ds*) =

(if *f* = *g* \wedge *length* *Cs* = *length* *Ds* then *concat* (*map* (*case-prod inf-mtxt-args*) (*zip* *Cs* *Ds*))

else [*MFun* *f* *Cs*]) |

inf-mtxt-args *C* *D* = [*C*]

lemma *inf-mtxt-args-MHole2* [*simp*]:

inf-mtxt-args *C* *MHole* = [*C*]

by (*cases* *C*) *simp-all*

lemma *fill-holes-mtxt-replicate-MHole* [*simp*]:

fill-holes-mtxt *C* (*replicate* (*num-holes* *C*) *MHole*) = *C*

proof (*induct* *C*)

case (*MFun* *f* *Cs*)

{ **fix** *i* **assume** *i* < *length* *Cs*

```

    then have partition-holes (replicate (sum-list (map num-holes Cs)) MHole) Cs
    ! i =
      replicate (num-holes (Cs ! i)) MHole
      using partition-by-nth-nth[of map num-holes Cs replicate (sum-list (map
num-holes Cs)) MHole]
      by (auto intro!: nth-equalityI)
    } note * = this
    show ?case using MFun[OF nth-mem] by (auto simp: * intro!: nth-equalityI)
qed auto

```

```

lemma num-holes-inf-mtxt:
  num-holes (C  $\sqcap$  D) = length (inf-mtxt-args C D)
  by (induct C D rule: inf-mtxt.induct)
    (auto simp: in-set-zip length-concat intro!: arg-cong [of - - sum-list])

```

```

lemma length-inf-mtxt-args:
  length (inf-mtxt-args D C) = length (inf-mtxt-args C D)
  by (metis inf commute num-holes-inf-mtxt)

```

```

lemma inf-mtxt-args-same [simp]:
  inf-mtxt-args C C = replicate (num-holes C) MHole
proof (induct C)
  case (MFun f Cs)
  have *:  $\bigwedge C. \text{num-holes } C = \text{length } (\text{inf-mtxt-args } C C)$ 
    using num-holes-inf-mtxt [of C C for C] by auto
  let ?xs = map (case-prod inf-mtxt-args) (zip Cs Cs)
  have  $\forall i < \text{length } Cs.$ 
    inf-mtxt-args (Cs ! i) (Cs ! i) = replicate (num-holes (Cs ! i)) MHole using
MFun by auto
  then have  $\forall i < \text{length } ?xs. \forall j < \text{length } (?xs ! i). ?xs ! i ! j = \text{MHole}$  by auto
  then have  $\forall i < \text{length } (\text{concat } ?xs). \text{concat } ?xs ! i = \text{MHole}$  by (metis nth-concat-two-lists)

  then show ?case by (auto simp: * intro!: nth-equalityI)
qed simp-all

```

```

lemma inf-mtxt-inf-mtxt-args:
  fill-holes-mtxt (C  $\sqcap$  D) (inf-mtxt-args C D) = C
proof (induct C D rule: inf-mtxt.induct)
  case ( $\lambda f Cs g Ds$ )
  then show ?case
  proof (cases  $f = g \wedge \text{length } Cs = \text{length } Ds$ )
    case True
    with  $\lambda$  have  $\forall i < \text{length } Cs.$ 
      fill-holes-mtxt (Cs ! i  $\sqcap$  Ds ! i) (inf-mtxt-args (Cs ! i) (Ds ! i)) = Cs ! i
    by (force simp: set-zip)
    moreover have partition-holes (concat (map (case-prod inf-mtxt-args) (zip Cs
Ds)))
      (map (case-prod ( $\sqcap$ )) (zip Cs Ds)) = map (case-prod inf-mtxt-args) (zip Cs
Ds)

```

```

    by (rule partition-by-concat-id) (simp-all add: num-holes-inf-mctxt)
  ultimately show ?thesis
    using fill-holes-mctxt.simps [simp del]
    by (auto simp: partition-holes-fill-holes-mctxt-conv intro!: nth-equalityI)
qed auto
qed auto

```

```

lemma inf-mctxt-inf-mctxt-args2:
  fill-holes-mctxt (C  $\sqcap$  D) (inf-mctxt-args D C) = D
  unfolding inf-mctxt-comm [of C D] by (rule inf-mctxt-inf-mctxt-args)

```

```

instantiation mctxt :: (type, type) sup
begin

```

```

fun sup-mctxt :: ('a, 'b) mctxt  $\Rightarrow$  ('a, 'b) mctxt  $\Rightarrow$  ('a, 'b) mctxt
where
  MHole  $\sqcap$  D = D |
  C  $\sqcap$  MHole = C |
  MVar x  $\sqcap$  MVar y = (if x = y then MVar x else undefined) |
  MFun f Cs  $\sqcap$  MFun g Ds =
    (if f = g  $\wedge$  length Cs = length Ds then MFun f (map (case-prod ( $\sqcap$ )) (zip Cs
Ds))
    else undefined) |
  (C :: ('a, 'b) mctxt)  $\sqcap$  D = undefined

```

```

instance ..

```

```

end

```

```

lemma sup-mctxt-idem [simp]:
  fixes C :: ('f, 'v) mctxt
  shows C  $\sqcap$  C = C
  by (induct C) (auto simp: zip-same-conv-map intro: map-idI)

```

```

lemma sup-mctxt-MHole [simp]: C  $\sqcap$  MHole = C
  by (induct C) simp-all

```

```

lemma sup-mctxt-comm [ac-simps]:
  fixes C :: ('f, 'v) mctxt
  shows C  $\sqcap$  D = D  $\sqcap$  C
  by (induct C D rule: sup-mctxt.induct) (fastforce simp: in-set-conv-nth intro!:
nth-equalityI)+

```

(\sqcap) is defined on compatible multihole-contexts. Note that compatibility is not transitive.

```

inductive-set comp-mctxt :: (('a, 'b) mctxt  $\times$  ('a, 'b) mctxt) set
where

```

```

  MHole1: (MHole, D)  $\in$  comp-mctxt |
  MHole2: (C, MHole)  $\in$  comp-mctxt |

```

$MVar: x = y \implies (MVar\ x, MVar\ y) \in comp_mctxt \mid$
 $MFun: f = g \implies length\ Cs = length\ Ds \implies \forall i < length\ Ds. (Cs\ !\ i, Ds\ !\ i) \in$
 $comp_mctxt \implies$
 $(MFun\ f\ Cs, MFun\ g\ Ds) \in comp_mctxt$

lemma *comp-mctxt-refl*:
 $(C, C) \in comp_mctxt$
by (*induct* C) (*auto intro: comp-mctxt.intros*)

lemma *comp-mctxt-sym*:
assumes $(C, D) \in comp_mctxt$
shows $(D, C) \in comp_mctxt$
using *assms* **by** (*induct*) (*auto intro: comp-mctxt.intros*)

lemma *sup-mctxt-assoc [ac-simps]*:
assumes $(C, D) \in comp_mctxt$ **and** $(D, E) \in comp_mctxt$
shows $C \sqcup D \sqcup E = C \sqcup (D \sqcup E)$
using *assms* **by** (*induct* $C\ D$ *arbitrary: E*) (*auto elim!: comp-mctxt.cases intro!: nth-equalityI*)

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* $(\leq) (<)$
by (*standard*) (*simp add: less-eq-mctxt-def*)

lemma *sup-mctxt-ge1 [simp]*:
assumes $(C, D) \in comp_mctxt$
shows $C \leq C \sqcup D$
using *assms* **by** (*induct* $C\ D$) (*auto simp: less-eq-mctxt-def intro: nth-equalityI*)

lemma *sup-mctxt-ge2 [simp]*:
assumes $(C, D) \in comp_mctxt$
shows $D \leq C \sqcup D$
using *assms* **by** (*induct*) (*auto simp: less-eq-mctxt-def intro: nth-equalityI*)

lemma *sup-mctxt-least*:
assumes $(D, E) \in comp_mctxt$
and $D \leq C$ **and** $E \leq C$
shows $D \sqcup E \leq C$
using *assms*
proof (*induct arbitrary: C*)
case $(MFun\ f\ g\ Cs\ Ds)$
then show *?case*
apply (*auto simp: less-eq-mctxt-def elim!: inf-mctxt.elims intro!: nth-equalityI*)[1]
apply (*metis (erased, lifting) length-map nth-map nth-zip split-conv*)
by (*metis mctxt.distinct(5)*)
qed *auto*

```

lemma inf-mtxt-args-MHole:
  assumes  $(C, D) \in \text{comp-mtxt}$  and  $i < \text{length} (\text{inf-mtxt-args } C \ D)$ 
  shows  $\text{inf-mtxt-args } C \ D \ ! \ i = \text{MHole} \vee \text{inf-mtxt-args } D \ C \ ! \ i = \text{MHole}$ 
using assms
proof (induct  $C \ D$  arbitrary:  $i$ )
  case (MHole2  $C$ )
  then show ?case by (cases  $C$ ) simp-all
next
  case (MFun  $f \ g \ Cs \ Ds$ )
  then have [simp]:  $f = g \ \text{length } Ds = \text{length } Cs$  by auto
  let  $?xs = \text{map} (\text{case-prod } \text{inf-mtxt-args}) (\text{zip } Cs \ Ds)$ 
  let  $?ys = \text{map} (\text{case-prod } \text{inf-mtxt-args}) (\text{zip } Ds \ Cs)$ 
  obtain  $m$  and  $n$  where  $*$ :  $i = \text{sum-list} (\text{map } \text{length} (\text{take } m \ ?xs)) + n$ 
   $m < \text{length } Cs \ n < \text{length} (\text{inf-mtxt-args } (Cs \ ! \ m) \ (Ds \ ! \ m))$ 
  and  $\text{inf-mtxt-args } (\text{MFun } f \ Cs) \ (\text{MFun } g \ Ds) \ ! \ i = \text{inf-mtxt-args } (Cs \ ! \ m) \ (Ds \ ! \ m) \ ! \ n$ 
  using MFun.prems by (auto dest: less-length-concat)
  moreover have  $\text{concat } ?ys \ ! \ i = (\text{map} (\text{case-prod } \text{inf-mtxt-args}) (\text{zip } Ds \ Cs)) \ ! \ m \ ! \ n$ 
  by (rule concat-nth)
  (insert  $*$ , auto intro: arg-cong [of - - sum-list]
  simp: map-nth-eq-conv length-inf-mtxt-args)
  ultimately show ?case using MFun(3) by simp
qed auto

```

Parallel rewriting is closed under multihole-contexts.

```

lemma par-rstep-mtxt:
  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  $(s, t) \in \text{par-rstep } R$ 
proof -
  have [simp]:  $\text{length } ss = \text{length } ts$  using assms by (auto dest!: eqfE)
  have [simp]:  $t = \text{fill-holes } C \ ts$  using assms by (auto dest: eqfE)
  have  $(s, \text{fill-holes } C \ ts) \in \text{par-rstep } R$ 
  using assms by (intro eqf-all-ctxt-closed-step [of UNIV - s C ss, THEN conjunct1]) auto
  then show ?thesis by simp
qed

```

```

lemma par-rstep-mtxtD:
  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  $\exists C \ ss \ ts. ?P \ s \ t \ C \ ss \ ts$ )
using assms
proof (induct)
  case (root-step  $s \ t \ \sigma$ )
  then have  $(s \cdot \sigma, t \cdot \sigma) \in \text{rrstep } R$  by auto

```

moreover have $s \cdot \sigma =_f (MHole, [s \cdot \sigma])$ and $t \cdot \sigma =_f (MHole, [t \cdot \sigma])$ by auto
 ultimately show ?case by force
 next
 case (par-step-var x)
 have $Var\ x =_f (MVar\ x, [])$ by auto
 then show ?case by force
 next
 case (par-step-fun ts ss f)
 then have $\forall i < \text{length}\ ts. \exists x. ?P\ (ss\ !\ i)\ (ts\ !\ i)\ (fst\ x)\ (fst\ (snd\ x))\ (snd\ (snd\ x))$ by force
 then obtain g where $\forall i < \text{length}\ ts. ?P\ (ss\ !\ i)\ (ts\ !\ i)\ (fst\ (g\ i))\ (fst\ (snd\ (g\ i)))\ (snd\ (snd\ (g\ i)))$
 unfolding choice-iff' by blast
 moreover
 define Cs us vs where $Cs = \text{map}\ (\lambda i. fst\ (g\ i))\ [0 ..< \text{length}\ ts]$
 and $us = \text{map}\ (\lambda i. fst\ (snd\ (g\ i)))\ [0 ..< \text{length}\ ts]$
 and $vs = \text{map}\ (\lambda i. snd\ (snd\ (g\ i)))\ [0 ..< \text{length}\ ts]$
 ultimately have [simp]: $\text{length}\ Cs = \text{length}\ ts$
 $\text{length}\ us = \text{length}\ ts$ $\text{length}\ vs = \text{length}\ ts$
 and *: $\forall i < \text{length}\ us. ss\ !\ i =_f (Cs\ !\ i, us\ !\ i) \wedge ts\ !\ i =_f (Cs\ !\ i, vs\ !\ i) \wedge$
 $(\forall j < \text{length}\ (us\ !\ i). (us\ !\ i\ !\ j, vs\ !\ i\ !\ j) \in \text{rrstep}\ R)$
 by simp-all
 define C where $C = MFun\ f\ Cs$
 have $Fun\ f\ ss =_f (C, \text{concat}\ us)$ and $Fun\ f\ ts =_f (C, \text{concat}\ vs)$
 using * by (auto simp: C-def ‹length ss = length ts› intro: eqf-MFunI)
 moreover have $\forall i < \text{length}\ (\text{concat}\ us). (\text{concat}\ us\ !\ i, \text{concat}\ vs\ !\ i) \in \text{rrstep}\ R$
 using * by (intro concat-all-nth) (auto dest!: eqfE)
 ultimately show ?case by blast
 qed

lemma rsteps-mctxt:

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 $(s, t) \in (\text{rstep}\ R)^*$
 proof –
 have [simp]: $\text{length}\ ss = \text{length}\ ts$ using assms by (auto dest!: eqfE)
 have [simp]: $t = \text{fill-holes}\ C\ ts$ using assms by (auto dest: eqfE)
 have $(s, \text{fill-holes}\ C\ ts) \in (\text{rstep}\ R)^*$
 using assms by (intro eqf-all-ctxt-closed-step [of UNIV - s C ss, THEN conjunct1]) auto
 then show ?thesis by simp
 qed

fun sup-mctxt-args :: $(f, 'v)\ mctxt \Rightarrow (f, 'v)\ mctxt \Rightarrow (f, 'v)\ mctxt\ list$
 where

$\text{sup-mctxt-args}\ MHole\ D = [D] \mid$
 $\text{sup-mctxt-args}\ C\ MHole = \text{replicate}\ (\text{num-holes}\ C)\ MHole \mid$
 $\text{sup-mctxt-args}\ (MVar\ x)\ (MVar\ y) = (\text{if}\ x = y\ \text{then}\ []\ \text{else}\ \text{undefined}) \mid$
 $\text{sup-mctxt-args}\ (MFun\ f\ Cs)\ (MFun\ g\ Ds) =$

$(\text{if } f = g \wedge \text{length } Cs = \text{length } Ds \text{ then concat } (\text{map } (\text{case-prod sup-mtxt-args})$
 $(\text{zip } Cs Ds))$
 $\text{else undefined}) \mid$
 $\text{sup-mtxt-args } C D = \text{undefined}$

lemma *sup-mtxt-args-MHole2* [simp]:
 $\text{sup-mtxt-args } C \text{ MHole} = \text{replicate } (\text{num-holes } C) \text{ MHole}$
by (cases *C*) simp-all

lemma *num-holes-sup-mtxt-args*:
assumes $(C, D) \in \text{comp-mtxt}$
shows $\text{num-holes } C = \text{length } (\text{sup-mtxt-args } C D)$
using *assms* **by** (induct) (auto simp: length-concat intro!: arg-cong [of - - sum-list]
nth-equalityI)

lemma *sup-mtxt-sup-mtxt-args*:
assumes $(C, D) \in \text{comp-mtxt}$
shows $\text{fill-holes-mtxt } C (\text{sup-mtxt-args } C D) = C \sqcup D$
using *assms*
proof (induct)
note *fill-holes-mtxt.simps* [simp del]
case (MFun *f g Cs Ds*)
then show ?case
proof (cases $f = g \wedge \text{length } Cs = \text{length } Ds$)
case True
with MFun **have** $\forall i < \text{length } Cs.$
 $\text{fill-holes-mtxt } (Cs ! i) (\text{sup-mtxt-args } (Cs ! i) (Ds ! i)) = Cs ! i \sqcup Ds ! i$
and *: $\forall i < \text{length } Cs. (Cs ! i, Ds ! i) \in \text{comp-mtxt}$ **by** (force simp: set-zip)+
moreover have $\text{partition-holes } (\text{concat } (\text{map } (\text{case-prod sup-mtxt-args}) (\text{zip } Cs Ds)))$
 $Cs = \text{map } (\text{case-prod sup-mtxt-args}) (\text{zip } Cs Ds)$
using True **and** * **by** (intro partition-by-concat-id) (auto simp: num-holes-sup-mtxt-args)
ultimately show ?thesis
using * **and** True **by** (auto simp: partition-holes-fill-holes-mtxt-conv intro!:
nth-equalityI)
qed auto
qed auto

lemma *sup-mtxt-args*:
assumes $(C, D) \in \text{comp-mtxt}$
shows $\text{sup-mtxt-args } C D = \text{inf-mtxt-args } (C \sqcup D) C$
using *assms* **by** (induct) (auto intro!: arg-cong [of - - concat] *nth-equalityI*)

lemma *term-for-mtxt*:
fixes $C :: ('f, 'v) \text{ mtxt}$
obtains *t* **and** *ts* **where** $t =_f (C, ts)$
proof –
obtain *ts* :: $('f, 'v) \text{ term list}$ **where** $\text{num-holes } C = \text{length } ts$ **by** (metis *Ex-list-of-length*)
then have $\text{fill-holes } C ts =_f (C, ts)$ **by** blast

```

  show ?thesis by (standard) fact
qed

lemma comp-mctxt-eqfE:
  assumes (C, D) ∈ comp-mctxt
  obtains s and ss and ts where s =f (C, ss) and s =f (D, ts)
proof (goal-cases)
  case 1
  obtain u and us where u =f (C ⊔ D, us) by (metis term-for-mctxt)
  then have u: u = fill-holes (C ⊔ D) us
    and *: length us = num-holes (C ⊔ D) by (auto dest: eqfE)
  define Cs Ds where Cs = sup-mctxt-args C D
    and Ds = sup-mctxt-args D C
  then have sup1: C ⊔ D = fill-holes-mctxt C Cs and sup2: C ⊔ D = fill-holes-mctxt
    D Ds
  using assms by (auto simp: sup-mctxt-sup-mctxt-args comp-mctxt-sym ac-simps)
  then have u1: u = fill-holes (fill-holes-mctxt C Cs) us
    and u2: u = fill-holes (fill-holes-mctxt D Ds) us by (simp-all add: u)
  define ss ts where ss = map (λi. fill-holes (Cs ! i) (partition-holes us Cs ! i))
    [0 ..< num-holes C]
    and ts = map (λi. fill-holes (Ds ! i) (partition-holes us Ds ! i)) [0 ..< num-holes
    D]
  have u = fill-holes C ss
    using assms
    by (simp add: * u1 sup1 ss-def fill-holes-mctxt-fill-holes Cs-def num-holes-sup-mctxt-args)
  moreover have u = fill-holes D ts
    using assms [THEN comp-mctxt-sym]
    by (simp add: * u2 sup2 ts-def fill-holes-mctxt-fill-holes Ds-def num-holes-sup-mctxt-args)
  ultimately have u =f (C, ss) and u =f (D, ts) by (auto simp: ss-def ts-def)
  from 1[OF this] show thesis .
qed

lemma eqf-comp-mctxt:
  assumes s =f (C, ss) and s =f (D, ts)
  shows (C, D) ∈ comp-mctxt
using assms
proof (induct s arbitrary: C D ss ts)
  case (Var x C D)
  then show ?case
    by (cases C D rule: mctxt.exhaust [case-product mctxt.exhaust])
      (auto simp: eq-fill.simps intro: comp-mctxt.intros)
next
  case (Fun f ss C D us vs)
  { fix Cs and Ds
    assume *: C = MFun f Cs D = MFun f Ds and **: length Cs = length Ds
    have ?case
    proof (unfold *, intro comp-mctxt.MFun [OF refl **]) all i impI
      fix i
      assume i < length Ds

```

```

    then show (Cs ! i, Ds ! i) ∈ comp-mctxt
    using Fun by (auto simp: * ** elim!: eqf-MFunE) (metis nth-mem)
  qed }
with Fun.premis show ?case
  by (cases C D rule: mctxt.exhaust [case-product mctxt.exhaust])
    (auto simp: eq-fill.simps dest: map-eq-imp-length-eq intro: comp-mctxt.intros)
qed

lemma comp-mctxt-iff:
  (C, D) ∈ comp-mctxt ⟷ (∃ s ss ts. s =f (C, ss) ∧ s =f (D, ts))
  by (blast elim!: comp-mctxt-eqfE intro: eqf-comp-mctxt)

lemma hole-poss-parallel-pos [simp]:
  assumes p ∈ hole-poss C and q ∈ hole-poss C and p ≠ q
  shows parallel-pos p q
using assms by (induct C arbitrary: p q) (fastforce dest!: nth-mem)+

lemma eq-fill-induct [consumes 1, case-names MHole MVar MFun]:
  assumes t =f (C, ts)
  and ∧t. P t MHole [t]
  and ∧x. P (Var x) (MVar x) []
  and ∧f ss Cs ts. [length Cs = length ss; sum-list (map num-holes Cs) = length
ts;
  ∀ i < length ss. ss ! i =f (Cs ! i, partition-holes ts Cs ! i) ∧
  P (ss ! i) (Cs ! i) (partition-holes ts Cs ! i)]
  ⇒ P (Fun f ss) (MFun f Cs) ts
  shows P t C ts
using assms(1)
proof (induct t arbitrary: C ts)
  case (Var x)
  then show ?case
    using assms(2, 3) by (cases C; cases ts) (auto elim: eq-fill.cases)
next
  case (Fun f ss C ts)
  { assume C = MHole and ts = [Fun f ss]
    with Fun.hyps have ?case using assms(2) by auto }
  moreover
  { fix Cs
    assume C: C = MFun f Cs and sum-list (map num-holes Cs) = length ts
    and length Cs = length ss
    and Fun f ss = fill-holes (MFun f Cs) ts
    moreover then have ∀ i < length ss. ss ! i =f (Cs ! i, partition-holes ts Cs !
i)
    by (auto simp del: fill-holes.simps
        simp: partition-holes-fill-holes-conv intro!: eq-fill.intros)
    (metis (no-types, lifting) add.left-neutral length-map length-upt nth-map-upt)
    moreover with Fun.hyps(1) have ∀ i < length ss.
    P (ss ! i) (Cs ! i) (partition-holes ts Cs ! i) by auto
    ultimately have ?case using assms(4) [of Cs ss ts f] by auto }

```

```

ultimately show ?case
  using Fun.premis by (elim eq-fill.cases) (auto, cases C; cases ts, auto)
qed

lemma hole-poss-subset-poss:
  assumes  $s =_f (C, ss)$ 
  shows  $\text{hole-poss } C \subseteq \text{poss } s$ 
using assms by (induct rule: eq-fill-induct) auto

fun hole-num
where
  hole-num [] MHole = 0 |
  hole-num (i # q) (MFun f Cs) = sum-list (map num-holes (take i Cs)) + hole-num
    q (Cs ! i)

lemma hole-poss-nth-subt-at:
  assumes  $t =_f (C, ts)$  and  $p \in \text{hole-poss } C$ 
  shows  $\text{hole-num } p \ C < \text{length } ts \wedge t \mid - p = ts ! \text{hole-num } p \ C$ 
using assms
proof (induct arbitrary: p rule: eq-fill-induct)
  case (MFun f ss Cs ts)
  let ?ts = partition-holes ts Cs
  from MFun obtain i and q where [simp]:  $p = i \# q$ 
    and  $i < \text{length } ss$  and  $q \in \text{hole-poss } (Cs ! i)$  by auto
  with MFun.hyps have  $ss ! i =_f (Cs ! i, ?ts ! i)$ 
    and  $j: \text{hole-num } q \ (Cs ! i) < \text{length } (?ts ! i)$  (is  $?j < \text{length } -$ )
    and  $*: ?ts ! i ! \text{hole-num } q \ (Cs ! i) = ss ! i \mid - q$ 
    by auto
  let ?k = sum-list (map length (take i ?ts)) + ?j
  have  $i < \text{length } ?ts$  using  $\langle i < \text{length } ss \rangle$  and MFun by auto
  with partition-by-nth-nth-old [OF this j] and MFun and concat-nth-length [OF
    this j]
    have  $?ts ! i ! ?j = ts ! ?k$  and  $?k < \text{length } ts$  by (auto)
  moreover with * have  $ts ! ?k = \text{Fun } f \ ss \mid - p$  using  $\langle i < \text{length } ss \rangle$  by simp
  ultimately show ?case using MFun.hyps(2) by (auto simp: take-map [symmetric])
qed auto

lemma eqf-Fun-MFun:
  assumes  $\text{Fun } f \ ss =_f (MFun \ g \ Cs, ts)$ 
  shows  $g = f \wedge \text{length } Cs = \text{length } ss \wedge \text{sum-list } (\text{map num-holes } Cs) = \text{length }
    ts \wedge$ 
     $(\forall i < \text{length } ss. ss ! i =_f (Cs ! i, \text{partition-holes } ts \ Cs ! i))$ 
using assms by (induct Fun f ss MFun g Cs ts rule: eq-fill-induct) auto

lemma fill-holes-eq-Var-cases:
  assumes  $\text{num-holes } C = \text{length } ts$ 
  and  $\text{fill-holes } C \ ts = \text{Var } x$ 
  obtains  $C = \text{MHole} \wedge ts = [\text{Var } x] \mid C = \text{MVar } x \wedge ts = []$ 
using assms by (induct C; cases ts) auto

```

```

lemma num-holes-inf-mtxt-le:
  assumes  $s =_f (C, ts)$  and  $s =_f (D, us)$ 
  shows  $\text{num-holes } (C \sqcap D) \leq \text{num-holes } C + \text{num-holes } D$ 
using assms
proof (induct C D arbitrary: s ts us rule: inf-mtxt.induct)
  case ( $\lambda f Cs g Ds$ )
  show ?case
  proof (cases f = g  $\wedge$  length Cs = length Ds)
    case False
    with  $\lambda$  show ?thesis by (auto elim!: eq-fill.cases dest!: map-eq-imp-length-eq)
  next
  case True
  then have [simp]:  $g = f$   $\text{length } Ds = \text{length } Cs$  by simp-all
  have IH:  $\forall (C, D) \in \text{set } (\text{zip } Cs Ds). \text{num-holes } (C \sqcap D) \leq \text{num-holes } C +$ 
 $\text{num-holes } D$ 
  proof
    fix  $C D$  assume *:  $(C, D) \in \text{set } (\text{zip } Cs Ds)$ 
    then obtain  $i$  where  $i < \text{length } Cs$  and  $\text{zip } Cs Ds ! i = (C, D)$  by (auto
 $\text{simp: in-set-zip}$ )
    with  $\lambda$ .prems
    have  $\text{fill-holes } (Cs ! i) (\text{partition-holes } ts \ Cs ! i) =_f (C, \text{partition-holes } ts$ 
 $Cs ! i)$ 
    and  $\text{fill-holes } (Cs ! i) (\text{partition-holes } ts \ Cs ! i) =_f (D, \text{partition-holes } us$ 
 $Ds ! i)$ 
    by (auto elim!: eq-fill.cases)
    from  $\lambda$ .hyps [OF True * HOL.refl this]
    show  $\text{num-holes } (C \sqcap D) \leq \text{num-holes } C + \text{num-holes } D$  .
  qed
  have  $\text{num-holes } (MFun f Cs \sqcap MFun g Ds) = \text{sum-list } (\text{map } (\text{num-holes } \circ$ 
 $\text{case-prod } (\sqcap)) (\text{zip } Cs Ds))$ 
  using  $\lambda$ .prems by (auto elim!: eq-fill.cases dest!: map-eq-imp-length-eq)
  moreover have  $\text{num-holes } (MFun f Cs) + \text{num-holes } (MFun g Ds) =$ 
 $\text{sum-list } (\text{map } (\lambda(C, D). \text{num-holes } C + \text{num-holes } D) (\text{zip } Cs Ds))$ 
  using  $\langle \text{length } Ds = \text{length } Cs \rangle$  by (induct rule: list-induct2) simp-all
  ultimately show ?thesis using IH by (auto intro!: sum-list-mono)
qed
qed (auto elim!: eq-fill.cases)

lemma map-inf-mtxt-zip-mtxt-of-term [simp]:
   $\text{map } (\lambda(x, y). x \sqcap y) (\text{zip } (\text{map mtxt-of-term } ts) (\text{map mtxt-of-term } ts)) = \text{map}$ 
 $\text{mtxt-of-term } ts$ 
by (induct ts) simp-all

lemma inf-mtxt-ctxt-apply-term [simp]:
   $\text{mtxt-of-term } (C\langle t \rangle) \sqcap \text{mtxt-of-ctxt } C = \text{mtxt-of-ctxt } C$ 
 $\text{mtxt-of-ctxt } C \sqcap \text{mtxt-of-term } (C\langle t \rangle) = \text{mtxt-of-ctxt } C$ 
by (induct C) simp-all

```

lemma *inf-fill-holes-mctxt-MHoles*:

num-holes C = length Cs \implies length Ds = length Cs \implies

$\forall i < \text{length } Cs. Cs ! i = \text{MHole} \vee Ds ! i = \text{MHole} \implies$

fill-holes-mctxt C Cs \sqcap fill-holes-mctxt C Ds = C

proof (*induct C arbitrary: Cs Ds*)

case (*MHole Cs Ds*)

then show *?case by (cases Cs; cases Ds; force)*

next

case (*MFun f Bs Cs Ds*)

then show *?case*

unfolding *partition-holes-fill-holes-mctxt-conv'*

apply *simp*

apply (*rule nth-equalityI*)

by (*auto simp: partition-by-nth-nth*)

qed *auto*

lemma *inf-fill-holes-mctxt-two-MHoles [simp]*: *num-holes C = 2 \implies*

fill-holes-mctxt C [MHole, D] \sqcap fill-holes-mctxt C [E, MHole] = C

by (*simp add: inf-fill-holes-mctxt-MHoles nth-Cons'*)

lemma *two-subterms-cases*:

assumes *s = C⟨t⟩ and s = D⟨u⟩*

obtains (*eq*) *C = D and t = u*

| (*nested1*) *C' where C' $\neq \square$ and C = D \circ_c C'*

| (*nested2*) *D' where D' $\neq \square$ and D = C \circ_c D'*

| (*parallel1*) *E where num-holes E = 2*

and *mctxt-of-ctxt C = fill-holes-mctxt E [MHole, mctxt-of-term u]*

and *mctxt-of-ctxt D = fill-holes-mctxt E [mctxt-of-term t, MHole]*

| (*parallel2*) *E where num-holes E = 2*

and *mctxt-of-ctxt C = fill-holes-mctxt E [mctxt-of-term u, MHole]*

and *mctxt-of-ctxt D = fill-holes-mctxt E [MHole, mctxt-of-term t]*

proof (*atomize-elim, insert assms, induct s arbitrary: C t D u*)

case (*Var x*)

then show *?case by (cases C; cases D; cases t; cases u) auto*

next

case (*Fun f ss*)

{ fix *ts₁ C' ts₂ and us₁ D' us₂*

assume [*simp*]: *C = More f ts₁ C' ts₂ D = More f us₁ D' us₂*

then have *len: length (ts₁ @ ts₂) + 1 = length ss length (us₁ @ us₂) + 1 =*

length ss

using *Fun.prem by (auto) (metis add-Suc-right length-Cons length-append nat.inject)*

{ assume *length ts₁ = length us₁*

with *Fun have* [*simp*]: *take (length ts₁) ss = ts₁ drop (Suc (length ts₁)) ss =*

ts₂

and [*simp*]: *us₁ = take (length ts₁) ss us₂ = drop (length ts₁ + 1) ss*

and *nth: C'⟨t⟩ = ss ! length ts₁ and mem: C'⟨t⟩ \in set ss*

and *eq: C'⟨t⟩ = D'⟨u⟩ by auto*

{ assume *C' = D' and t = u*

```

    then have  $C = D$  and  $t = u$  by simp-all
    then have ?case by blast }
  moreover
  { fix  $C''$  assume  $C'' \neq \square$  and  $C' = D' \circ_c C''$ 
    then have  $C'' \neq \square$  and  $C = D \circ_c C''$  by auto
    then have ?case by blast }
  moreover
  { fix  $D''$  assume  $D'' \neq \square$  and  $D' = C' \circ_c D''$ 
    then have  $D'' \neq \square$  and  $D = C \circ_c D''$  by auto
    then have ?case by blast }
  moreover
  { fix  $E'$  assume [simp]:  $\text{mctxt-of-ctxt } C' = \text{fill-holes-mctxt } E' [\text{MHole}, \text{mctxt-of-term } u]$ 
     $\text{mctxt-of-ctxt } D' = \text{fill-holes-mctxt } E' [\text{mctxt-of-term } t, \text{MHole}]$ 
     $\text{num-holes } E' = 2$ 
    define  $E$  where  $E = \text{MFun } f (\text{map } \text{mctxt-of-term } ts_1 @ E' \# \text{map } \text{mctxt-of-term } ts_2)$ 
    then have  $\text{num-holes } E = 2$  by simp
    moreover have  $\text{mctxt-of-ctxt } C = \text{fill-holes-mctxt } E [\text{MHole}, \text{mctxt-of-term } u]$ 
      unfolding  $E\text{-def}$  and partition-holes-fill-holes-mctxt-conv' by simp
    moreover have  $\text{mctxt-of-ctxt } D = \text{fill-holes-mctxt } E [\text{mctxt-of-term } t, \text{MHole}]$ 
      unfolding  $E\text{-def}$  and partition-holes-fill-holes-mctxt-conv' by simp
    ultimately have ?case by blast }
  moreover
  { fix  $E'$  assume [simp]:  $\text{mctxt-of-ctxt } C' = \text{fill-holes-mctxt } E' [\text{mctxt-of-term } u, \text{MHole}]$ 
     $\text{mctxt-of-ctxt } D' = \text{fill-holes-mctxt } E' [\text{MHole}, \text{mctxt-of-term } t]$ 
     $\text{num-holes } E' = 2$ 
    define  $E$  where  $E = \text{MFun } f (\text{map } \text{mctxt-of-term } ts_1 @ E' \# \text{map } \text{mctxt-of-term } ts_2)$ 
    then have  $\text{num-holes } E = 2$  by simp
    moreover have  $\text{mctxt-of-ctxt } C = \text{fill-holes-mctxt } E [\text{mctxt-of-term } u, \text{MHole}]$ 
      unfolding  $E\text{-def}$  and partition-holes-fill-holes-mctxt-conv' by simp
    moreover have  $\text{mctxt-of-ctxt } D = \text{fill-holes-mctxt } E [\text{MHole}, \text{mctxt-of-term } t]$ 
      unfolding  $E\text{-def}$  and partition-holes-fill-holes-mctxt-conv' by simp
    ultimately have ?case by blast }
  ultimately have ?case using Fun.hyps [OF mem HOL.refl eq] by blast }
  moreover
  { assume *:  $\text{length } ts_1 < \text{length } us_1$ 
    moreover then have  $us_1: us_1 = ts_1 @ C'\langle t \rangle \# \text{drop } (\text{length } ts_1 + 1) us_1$ 
      using Fun.prems [simplified]
    apply (subst append-take-drop-id [symmetric, of - length ts1])
    apply (rule arg-cong2 [where  $f = (@)$ ])
    apply (force simp: append-eq-append-conv-if)
    apply (simp add: append-eq-append-conv-if)
    apply (cases us1)
  }

```

```

    by auto
      (metis Cons-eq-appendI Cons-nth-drop-Suc calculation drop-Suc-Cons
nth-append-length)
    ultimately have ss: ss = ts1 @ C'⟨t⟩ # drop (length ts1 + 1) us1 @ D'⟨u⟩
# us2
      using Fun.premis(2, 1) by auto
      have ts2: ts2 = drop (length ts1 + 1) us1 @ D'⟨u⟩ # us2
      using Fun.premis (2, 1) [simplified] and *
      apply (subst append-take-drop-id [symmetric, of - length (drop (length ts1
+ 1) us1)])
      apply (rule arg-cong2 [where f = (@)])
      by auto (metis Suc-eq-plus1 append-eq-conv-conj length-drop list.inject ss)+
      define E where E = MFun f (map mctxt-of-term ts1 @ mctxt-of-ctxt C' #
        map mctxt-of-term (drop (length ts1 + 1) us1) @ mctxt-of-ctxt D' # map
mctxt-of-term us2)
      then have num-holes E = 2 by simp
      moreover have mctxt-of-ctxt C = fill-holes-mctxt E [MHole, mctxt-of-term
u]
      unfolding E-def and partition-holes-fill-holes-mctxt-conv' by (simp add: *
ts2)
      moreover have mctxt-of-ctxt D = fill-holes-mctxt E [mctxt-of-term t, MHole]
      unfolding E-def and partition-holes-fill-holes-mctxt-conv' by (simp, subst
us1, simp)
      ultimately have ?case by blast }
    moreover
    { assume *: length us1 < length ts1
      moreover then have ts1: ts1 = us1 @ D'⟨u⟩ # drop (length us1 + 1) ts1
      using Fun.premis [simplified]
      apply (subst append-take-drop-id [symmetric, of - length us1])
      apply (rule arg-cong2 [where f = (@)])
      apply (force simp: append-eq-append-conv-if)
      apply (simp add: append-eq-append-conv-if)
      apply (cases ts1)
      by auto (metis Cons-eq-appendI Cons-nth-drop-Suc calculation drop-Suc-Cons
nth-append-length)
      ultimately have ss: ss = us1 @ D'⟨u⟩ # drop (length us1 + 1) ts1 @ C'⟨t⟩
# ts2
      using Fun.premis by auto
      have us2: us2 = drop (length us1 + 1) ts1 @ C'⟨t⟩ # ts2
      using Fun.premis (2, 1) [simplified] and *
      apply (subst append-take-drop-id [symmetric, of - length (drop (length us1
+ 1) ts1)])
      apply (rule arg-cong2 [where f = (@)])
      by auto (metis Suc-eq-plus1 append-eq-conv-conj length-drop list.inject ss)+
      define E where E = MFun f (map mctxt-of-term us1 @ mctxt-of-ctxt D' #
        map mctxt-of-term (drop (length us1 + 1) ts1) @ mctxt-of-ctxt C' # map
mctxt-of-term ts2)
      then have num-holes E = 2 by simp
      moreover have mctxt-of-ctxt C = fill-holes-mctxt E [mctxt-of-term u, MHole]

```

unfolding $E\text{-def}$ **and** $\text{partition-holes-fill-holes-mctxt-conv}'$ **by** (simp , $\text{subst } ts_1$, simp)
moreover have $\text{mctxt-of-ctxt } D = \text{fill-holes-mctxt } E \text{ [MHole, mctxt-of-term } t]$
unfolding $E\text{-def}$ **and** $\text{partition-holes-fill-holes-mctxt-conv}'$ **by** ($\text{simp add: } *$ us_2)
ultimately have $?case$ **by** blast }
moreover
have $\text{length } ts_1 = \text{length } us_1 \vee \text{length } ts_1 < \text{length } us_1 \vee \text{length } us_1 < \text{length } ts_1$ **by** arith
ultimately have $?case$ **by** blast }
moreover
{ assume } $C = \square$ **and $D \neq \square$ **then have** $?case$ **by** auto }
moreover
{ assume } $C \neq \square$ **and $D = \square$ **then have** $?case$ **by** auto }
moreover
{ assume } $C = \square$ **and $D = \square$ **then have** $?case$ **using** Fun **by** simp }
ultimately show $?case$ **using** Fun **by** ($\text{cases } C$; $\text{cases } D$) simp-all
qed******

lemma $\text{two-hole-ctxt-inf-conv}$:

$\text{num-holes } E = 2 \implies$
 $\text{mctxt-of-ctxt } C = \text{fill-holes-mctxt } E \text{ [MHole, mctxt-of-term } u] \implies$
 $\text{mctxt-of-ctxt } D = \text{fill-holes-mctxt } E \text{ [mctxt-of-term } t, \text{MHole}] \implies$
 $\text{mctxt-of-ctxt } C \sqcap \text{mctxt-of-ctxt } D = E$
by simp

lemma $\text{map-length-take-partition-by}$:

$i < \text{length } ys \implies \text{sum-list } ys = \text{length } xs \implies$
 $\text{map length (take } i \text{ (partition-by } xs \text{ } ys)) = \text{take } i \text{ } ys$
by ($\text{metis map-length-partition-by take-map}$)

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 $(\text{fill-holes } C (ss_1 @ t \# ss_2), \text{fill-holes } C (ss_1 @ u \# ss_2)) \in R$
using assms
proof ($\text{induct } C$ $\text{arbitrary: } ss_1 \ ss_2$)
case ($\text{MFun } f \ Cs$)
let $?f = \lambda x. \text{partition-holes } (ss_1 @ x \# ss_2) \ Cs$
let $?ts = ?f \ t$ **and** $?us = ?f \ u$
have $*$: $\bigwedge x. \text{concat } (?f \ x) = ss_1 @ x \# ss_2$
using MFun.premis **by** ($\text{intro concat-partition-by}$) simp
with $\text{less-length-concat}$ [$\text{of length } ss_1 \ ?ts$]
obtain $i \ j$ **where** ij : $\text{sum-list (map length (take } i \ ?ts)) + j = \text{length } ss_1$
 $i < \text{length } Cs \ j < \text{length } (?ts \ ! \ i)$
and [simp]: $?ts \ ! \ i \ ! \ j = t$ **by** auto

```

have length ss1 = sum-list (map length (take i ?us)) + j
  using ij using MFun.prems(3) by (auto simp: take-map [symmetric])
from concat-nth [OF - - this]
  have [simp]: ?us ! i ! j = u using ij and MFun.prems(3) by auto
have [simp]: length ?us = length ?ts by simp
have [simp]: take j (?us ! i) = take j (?ts ! i)
  drop (Suc j) (?us ! i) = drop (Suc j) (?ts ! i)
  using ij and MFun.prems(3)
  by (auto intro: nth-equalityI simp: nth-append concat-nth [symmetric] take-map
[symmetric])
from MFun.hyps [of Cs ! i, OF - MFun.prems(1, 2), of take j (?ts ! i) drop (Suc
j) (?ts ! i)]
  have step: (fill-holes (Cs ! i) (?ts ! i), fill-holes (Cs ! i) (?us ! i)) ∈ R
  using ij and MFun.prems
  apply simp
  apply (subst id-take-nth-drop [of j ?ts ! i])
  apply simp
  apply (subst id-take-nth-drop [of j ?us ! i])
  apply auto
  done

let ?Cs = map (case-prod fill-holes) (zip Cs ?ts)
let ?C = More f (take i ?Cs) □ (drop (Suc i) ?Cs)
have [simp]:
  take i (map (case-prod fill-holes) (zip Cs ?us)) = take i (map (case-prod fill-holes)
(zip Cs ?ts))
  drop (Suc i) (map (case-prod fill-holes) (zip Cs ?us)) = drop (Suc i) (map
(case-prod fill-holes) (zip Cs ?ts))
  using ij and MFun.prems(3)
  apply (auto intro!: nth-equalityI)[2]
  subgoal
    using partition-by-nth-less [of - i map num-holes Cs ss1 j - ss2]
    by (simp add: map-length-take-partition-by)
  subgoal using partition-by-nth-greater [of i Suc (i + k) for k, of - map
num-holes Cs j ss1 - ss2]
    by (simp add: map-length-take-partition-by)
  done
show ?case
  using MFun.prems(1) [rule-format, OF step, of ?C] and ij
  apply (clarsimp simp del: fill-holes.simps simp: partition-holes-fill-holes-conv')
  apply (subst id-take-nth-drop [of i map (case-prod fill-holes) (zip Cs ?ts)], simp)
  apply (subst id-take-nth-drop [of i map (case-prod fill-holes) (zip Cs ?us)], simp)
  by auto
qed auto

lemma mctxt-of-term-fill-holes':
  num-holes C = length ts ⇒ mctxt-of-term (fill-holes C ts) = fill-holes-mctxt C
(map mctxt-of-term ts)
  by (induct C ts rule: fill-holes-induct) auto

```

```

lemma vars-term-fill-holes':
  num-holes C = length ts  $\implies$  vars-term (fill-holes C ts) =  $\bigcup$  (vars-term ' set ts)
 $\bigcup$  vars-mtxt C
proof (induct C ts rule: fill-holes-induct)
  case (MFun f Cs ts) then show ?case
    using UN-upt-len-conv[of partition-holes ts Cs length Cs  $\lambda t$ . ( $\bigcup_{x \in \text{set } t}$  vars-term x)]
    by (simp add: UN-Un-distrib UN-set-partition-by)
qed auto

lemma vars-mtxt-linear: assumes t =f (C, ts)
  linear-term t
shows vars-mtxt C  $\cap \bigcup$  (vars-term ' set ts) = {}
  using assms
proof (induct C arbitrary: t ts)
  case (MVar x)
    from eqf-MVarE[OF MVar(1)]
    show ?case by auto
next
  case MHole
    from eqf-MHoleE[OF MHole(1)]
    show ?case by auto
next
  case (MFun f Cs t ss)
    from eqf-MFunE[OF MFun(2)] obtain ts sss where
      *: t = Fun f ts length ts = length Cs length sss = length Cs
       $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$ 
      ss = concat sss by blast
    {
      fix i
      assume i: i < length Cs
      hence mem: Cs ! i  $\in$  set Cs by auto
      from * i MFun(3) have lin: linear-term (ts ! i) by auto
      from MFun(1)[OF mem *(4)[OF i] lin]
      have vars-mtxt (Cs ! i)  $\cap \bigcup$  (vars-term ' set (sss ! i)) = {} by auto
    } note IH = this
    show ?case
    proof (rule ccontr)
      assume  $\neg$  ?thesis
      then obtain x where xC: x  $\in$  vars-mtxt (MFun f Cs) and xss: x  $\in \bigcup$ 
        (vars-term ' set ss)
      by auto
      from xC obtain i where i: i < length Cs and x: x  $\in$  vars-mtxt (Cs ! i)
      by (auto simp: set-conv-nth)
      from IH[OF i] x have xni: x  $\notin \bigcup$  (vars-term ' set (sss ! i)) by auto
      from *(4)[OF i] have ts ! i =f (Cs ! i, sss ! i) .
      from eqfE[OF this] x have xi: x  $\in$  vars-term (ts ! i)
      by (simp add: vars-term-fill-holes')

```

```

from  $xss[unfolded * set\text{-}concat]$  * obtain  $j$  where
   $j: j < length\ Cs$  and  $xsj: x \in \bigcup (vars\text{-}term\ 'set\ (sss\ !\ j))$ 
  unfolding  $set\text{-}conv\text{-}nth$  by auto
from  $*(4)[OF\ j]$  have  $ts\ !\ j =_f (Cs\ !\ j, sss\ !\ j)$  by auto
from  $eqfE[OF\ this]\ xsj\ j$  have  $xj: x \in vars\text{-}term\ (ts\ !\ j)$ 
  by (simp add: vars-term-fill-holes')
from  $xi\ xj\ i\ j\ \langle linear\text{-}term\ t \rangle[unfolded\ *(1)]$ 
have  $i = j$  unfolding  $\langle length\ ts = length\ Cs \rangle[symmetric]$ 
  by (auto simp: is-partition-alt is-partition-alt-def)
with  $xni\ xsj$  show False by auto
qed
qed

lemma mctxt-of-term-var-subst:
   $mctxt\text{-}of\text{-}term\ (t \cdot (Var \circ f)) = map\text{-}vars\text{-}mctxt\ f\ (mctxt\text{-}of\text{-}term\ t)$ 
by (induct t) auto

lemma subst-apply-mctxt-map-vars-mctxt-conv:
   $C \cdot mc\ (Var \circ f) = map\text{-}vars\text{-}mctxt\ f\ C$ 
by (induct C) auto

lemma map-vars-mctxt-mono:
   $C \leq D \implies map\text{-}vars\text{-}mctxt\ f\ C \leq map\text{-}vars\text{-}mctxt\ f\ D$ 
by (induct C D rule: less-eq-mctxt-induct) (auto intro: less-eq-mctxtI1)

lemma map-vars-mctxt-less-eq-decomp:
  assumes  $C \leq map\text{-}vars\text{-}mctxt\ f\ D$ 
  obtains  $C'$  where  $map\text{-}vars\text{-}mctxt\ f\ C' = C\ C' \leq D$ 
  using assms
proof (induct D arbitrary: C thesis)
  case ( $MVar\ x$ ) show ?case using  $MVar(1)[of\ MHole]\ MVar(1)[of\ MVar\ -]\ MVar(2)$ 
  by (auto elim: less-eq-mctxtE2 intro: less-eq-mctxtI1)
next
  case  $MHole$  show ?case using  $MHole(1)[of\ MHole]\ MHole(2)$  by (auto elim: less-eq-mctxtE2)
next
  case ( $MFun\ g\ Ds$ ) note  $MFun' = MFun$ 
  show ?case using  $MFun(3)$  unfolding  $map\text{-}vars\text{-}mctxt.simps$ 
  proof (cases rule: less-eq-mctxtE2(3))
  case  $MHole$  then show ?thesis using  $MFun(2)[of\ MHole]$  by auto
next
  case ( $MFun\ Cs$ )
  define  $Cs'$  where  $Cs' = map\ (\lambda i. SOME\ Ci'. map\text{-}vars\text{-}mctxt\ f\ Ci' = Cs\ !\ i \wedge Ci' \leq Ds\ !\ i)\ [0..<length\ Cs]$ 
  { fix  $i$  assume  $i < length\ Cs$ 
    obtain  $Ci'$  where  $map\text{-}vars\text{-}mctxt\ f\ Ci' = Cs\ !\ i\ Ci' \leq Ds\ !\ i$ 
    using  $\langle i < length\ Cs \rangle\ MFun\ MFun'(1)[OF\ nth\text{-}mem,\ of\ i]\ MFun'(3)$  by (auto elim!: less-eq-mctxtE2)
  }
```

```

    then have  $\exists Ci'. \text{map-vars-mctxt } f \text{ } Ci' = Cs ! i \wedge Ci' \leq Ds ! i$  by blast
  }
  from someI-ex[OF this] have
    length Cs = length Cs' and  $i < \text{length } Cs \implies \text{map-vars-mctxt } f \text{ } (Cs' ! i) =$ 
    Cs ! i
    i < length Cs  $\implies Cs' ! i \leq Ds ! i$  for i by (auto simp: Cs'-def)
  then show ?thesis using MFun(1,2) MFun'(3)
    by (auto intro!: MFun'(2)[of MFun g Cs] nth-equalityI less-eq-mctxtI2 elim!:
    less-eq-mctxtE2)
  qed
  qed

```

7.4 All positions of a multi-hole context

fun *all-poss-mctxt* :: (*'f*, *'v*) *mctxt* \Rightarrow *pos set*

where

```

  all-poss-mctxt (MVar x) = {}
| all-poss-mctxt MHole = {}
| all-poss-mctxt (MFun f cs) = {}  $\cup \bigcup (\text{set } (\text{map } (\lambda i. (\lambda p. i \# p)) \text{ 'all-poss-mctxt$ 
  (cs ! i)) [0 ..< length cs]))

```

lemma *all-poss-mctxt-simp* [simp]:

```

  all-poss-mctxt (MFun f cs) = {}  $\cup \{i \# p \mid i p. i < \text{length } cs \wedge p \in \text{all-poss-mctxt}$ 
  (cs ! i))
  by auto

```

declare *all-poss-mctxt.simps*(3)[simp del]

lemma *all-poss-mctxt-conv*:

```

  all-poss-mctxt C = poss-mctxt C  $\cup$  hole-poss C
  by (induct C) auto

```

lemma *root-in-all-poss-mctxt*[simp]:

```

  []  $\in$  all-poss-mctxt C
  by (cases C) auto

```

lemma *hole-poss-mctxt-of-term*[simp]:

```

  hole-poss (mctxt-of-term t) = {}
  by (induct t) auto

```

lemma *poss-mctxt-mctxt-of-term*[simp]:

```

  poss-mctxt (mctxt-of-term t) = poss t
  by (induct t) auto

```

lemma *hole-poss-subst*: *hole-poss* (*C* · *mc* σ) = *hole-poss* *C*

by (induct *C*, auto)

lemma *all-poss-mctxt-mctxt-of-term*[simp]:

```

all-poss-mctxt (mctxt-of-term t) = poss t
by (induct t) auto

lemma mctxt-of-term-leq-imp-eq:
  mctxt-of-term t ≤ C ⟷ mctxt-of-term t = C
  by (induct t arbitrary: C) (auto elim!: less-eq-mctxtE1 simp: map-nth-eq-conv)

lemma mctxt-of-term-inj:
  mctxt-of-term s = mctxt-of-term t ⟷ s = t
proof (induct s arbitrary: t)
  case (Var x t)
  show ?case by (cases t, auto)
next
  case (Fun f ss t)
  thus ?case by (cases t, auto simp: map-eq-conv' intro: nth-equalityI)
qed

lemma all-poss-mctxt-map-vars-mctxt [simp]:
  all-poss-mctxt (map-vars-mctxt f C) = all-poss-mctxt C
  by (induct C) auto

lemma fill-holes-mctxt-extends-all-poss:
  assumes length Ds = num-holes C shows all-poss-mctxt C ⊆ all-poss-mctxt
    (fill-holes-mctxt C Ds)
  using assms[symmetric] by (induct C Ds rule: fill-holes-induct) force+

lemma eqF-substD:
  assumes t · σ =f (C, ss)
  hole-poss C ⊆ poss t
  shows ∃ D ts. t =f (D, ts) ∧ C = D · mc σ ∧ ss = map (λ ti. ti · σ) ts
  using assms
proof (induct C arbitrary: t ss)
  case (MVar x t ss)
  from eqfE[OF MVar(1)] obtain y where t = Var y σ y = Var x ss = [] by
    (cases t, auto)
  thus ?case using MVar by (auto intro!: exI[of - MVar y])
next
  case (MHole t ss)
  from eqfE[OF MHole(1)]
  show ?case by (cases ss, auto intro!: exI[of - MHole] exI[of - [t]])
next
  case (MFun f Cs t ss)
  show ?case
  proof (cases is-Fun t)
    case True
    from eqf-MFunE[OF MFun(2)] obtain tss sss where
      tsigma: t · σ = Fun f tss and len: length tss = length Cs length sss = length
        Cs

```

```

    and args:  $\bigwedge i. i < \text{length } Cs \implies tss ! i =_f (Cs ! i, sss ! i)$ 
    and ss:  $ss = \text{concat } sss$  by auto
  from True tsigma obtain ts where  $t: t = \text{Fun } f \text{ } ts$  by (cases t, auto)
  from tsigma[unfolded t] have ts:  $tss = \text{map } (\lambda t. t \cdot \sigma) \text{ } ts$  by auto
  from len ts have length ts = length Cs by auto
  note len = this len
  {
    fix i
    assume  $i: i < \text{length } Cs$ 
    hence  $Cs ! i \in \text{set } Cs$  by auto
    note  $IH = \text{MFun}(1)[\text{OF this}]$ 
    from ts len i have  $ts ! i \cdot \sigma = tss ! i$  by auto
    also have  $\dots =_f (Cs ! i, sss ! i)$  using args[OF i] .
    finally have  $ts ! i \cdot \sigma =_f (Cs ! i, sss ! i)$  .
    note  $IH = IH[\text{OF this}]$ 
    from MFun(3)[unfolded t] i len
    have hole-poss (Cs ! i)  $\subseteq$  poss (ts ! i) by auto
    note  $IH = IH[\text{OF this}]$ 
  }
  hence  $\forall i. \exists D \text{ } tsi. i < \text{length } Cs \longrightarrow ts ! i =_f (D, tsi) \wedge Cs ! i = D \cdot \text{mc } \sigma$ 
 $\wedge sss ! i = \text{map } (\lambda ti. ti \cdot \sigma) \text{ } tsi$  by blast
  from choice[OF this] obtain D where
     $\forall i. \exists tsi. i < \text{length } Cs \longrightarrow ts ! i =_f (D \text{ } i, tsi) \wedge Cs ! i = D \text{ } i \cdot \text{mc } \sigma \wedge sss$ 
 $! i = \text{map } (\lambda ti. ti \cdot \sigma) \text{ } tsi$  ..
  from choice[OF this] obtain tsi where
     $IH: i < \text{length } Cs \implies ts ! i =_f (D \text{ } i, tsi \text{ } i) \wedge Cs ! i = D \text{ } i \cdot \text{mc } \sigma \wedge sss ! i =$ 
 $\text{map } (\lambda ti. ti \cdot \sigma) \text{ } (tsi \text{ } i)$  for i
    by auto
  let ?n = [0 ..< length Cs]
  show ?thesis
  proof (rule exI[of - MFun f (map D ?n)], rule exI[of - concat (map tsi ?n)],
    intro conjI)
    show  $\text{MFun } f \text{ } Cs = \text{MFun } f \text{ } (\text{map } D \text{ } ?n) \cdot \text{mc } \sigma$  using IH by (auto intro:
    nth-equalityI)
    show  $ss = \text{map } (\lambda ti. ti \cdot \sigma) \text{ } (\text{concat } (\text{map } tsi \text{ } ?n))$  unfolding ss
    using len(3) IH unfolding map-concat map-map o-def
    by (intro arg-cong[of - - concat], intro nth-equalityI, auto)
    show  $t =_f (\text{MFun } f \text{ } (\text{map } D \text{ } ?n), \text{concat } (\text{map } tsi \text{ } ?n))$  unfolding t
    by (intro eqf-MFunI, insert len IH, auto)
  qed
next
case False
then obtain x where  $t: t = \text{Var } x$  by auto
with MFun(3) have hole-poss (MFun f Cs) = {} by auto
hence num: num-holes (MFun f Cs) = 0 using hole-poss-empty-iff-num-holes-0
by blast
with eqfE[OF MFun(2)] t have ss:  $ss = [] \text{ } \sigma \text{ } x = \text{fill-holes } (\text{MFun } f \text{ } Cs) \text{ } []$  by
auto
show ?thesis unfolding t ss

```

```

proof (intro exI[of - MVar x] exI[of - Nil] conjI)
  have MVar x ·mc σ = mctxt-of-term (fill-holes (MFun f Cs) []) using ss by
simp
  also have ... = MFun f Cs using num
  by (metis mctxt-of-term-fill-holes mctxt-of-term-term-of-mctxt-id)
  finally show MFun f Cs = MVar x ·mc σ ..
qed auto
qed
qed

```

7.5 More operations on multi-hole contexts

```

fun root-mctxt :: ('f, 'v) mctxt ⇒ ('f × nat) option where
  root-mctxt MHole = None
| root-mctxt (MVar x) = None
| root-mctxt (MFun f Cs) = Some (f, length Cs)

```

```

fun mreplace-at :: ('f, 'v) mctxt ⇒ pos ⇒ ('f, 'v) mctxt ⇒ ('f, 'v) mctxt where
  mreplace-at C [] D = D
| mreplace-at (MFun f Cs) (i # p) D = MFun f (take i Cs @ mreplace-at (Cs ! i)
p D # drop (i+1) Cs)

```

```

fun subm-at :: ('f, 'v) mctxt ⇒ pos ⇒ ('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 ∈ hole-poss C ⇒ subm-at C p = MHole
by (induct C arbitrary: p) auto

```

```

lemma subm-at-mctxt-of-term:
  p ∈ poss t ⇒ subm-at (mctxt-of-term t) p = mctxt-of-term (subt-at t p)
by (induct t arbitrary: p) auto

```

```

lemma subm-at-mreplace-at[simp]:
  p ∈ all-poss-mctxt C ⇒ subm-at (mreplace-at C p D) p = D
by (induct C arbitrary: p) (auto simp: nth-append-take)

```

```

lemma replace-at-subm-at[simp]:
  p ∈ all-poss-mctxt C ⇒ mreplace-at C p (subm-at C p) = C
by (induct C arbitrary: p) (auto simp: id-take-nth-drop[symmetric])

```

```

lemma all-poss-mctxt-mreplace-atI1:
  p ∈ all-poss-mctxt C ⇒ q ∈ all-poss-mctxt C ⇒ ¬(p <_p q) ⇒ q ∈ all-poss-mctxt
(mreplace-at C p D)

```

```

proof (induct C arbitrary: p q)
  let ?hd = λp. (case p :: pos of i # - ⇒ i)
  case (MFun f Cs) then show ?case

```

by (cases ?hd p = ?hd q) (auto simp: nth-append-take less-pos-def nth-append-drop-is-nth-conv
nth-append-take-drop-is-nth-conv)

qed auto

lemma funas-mctxt-sup-mctxt:

(C, D) ∈ comp-mctxt ⇒ funas-mctxt (C ⊔ D) = funas-mctxt C ∪ funas-mctxt D

by (induct C D rule: comp-mctxt.induct) (auto simp: zip-nth-conv Un-Union-image)

lemma mctxt-of-term-not-hole [simp]:

mctxt-of-term t ≠ MHole

by (cases t) auto

lemma funas-mctxt-mctxt-of-term [simp]:

funas-mctxt (mctxt-of-term t) = funas-term t

by (induct t) auto

lemma funas-mctxt-mreplace-at:

assumes p ∈ all-poss-mctxt C

shows funas-mctxt (mreplace-at C p D) ⊆ funas-mctxt C ∪ funas-mctxt D

using assms

proof (induct C p D rule: mreplace-at.induct)

case (2 f Cs i p D) **then have** Cs: Cs = take i Cs @ Cs ! i # drop (Suc i) Cs

by (auto simp: id-take-nth-drop)

show ?case **using** 2 **by** (subst (2) Cs) auto

qed auto

lemma funas-mctxt-mreplace-at-hole:

assumes p ∈ hole-poss C

shows funas-mctxt (mreplace-at C p D) = funas-mctxt C ∪ funas-mctxt D (is
?L = ?R)

proof

show ?R ⊆ ?L **using** assms

proof (induct C p D rule: mreplace-at.induct)

case (1 C D) **then show** ?case **by** (cases C) auto

next

case (2 f Cs i p D) **then have** Cs: Cs = take i Cs @ Cs ! i # drop (Suc i) Cs

by (auto simp: id-take-nth-drop)

show ?case **using** 2 **by** (subst (1) Cs) auto

qed auto

next

show ?L ⊆ ?R **using** assms **by** (auto simp: all-poss-mctxt-conv funas-mctxt-mreplace-at)

qed

lemma map-vars-mctxt-fill-holes-mctxt:

assumes num-holes C = length Cs

shows map-vars-mctxt f (fill-holes-mctxt C Cs) = fill-holes-mctxt (map-vars-mctxt
f C) (map (map-vars-mctxt f) Cs)

using assms **by** (induct C Cs rule: fill-holes-induct) (auto simp: comp-def)

```

lemma map-vars-mctxt-map-vars-mctxt[simp]:
  shows map-vars-mctxt f (map-vars-mctxt g C) = map-vars-mctxt (f ∘ g) C
  by (induct C) auto

lemma funas-mctxt-fill-holes:
  assumes num-holes C = length ts
  shows funas-term (fill-holes C ts) = funas-mctxt C ∪ ⋃ (set (map funas-term ts))
  using funas-term-fill-holes-iff[OF assms] by auto

lemma mctxt-neq-mholeE:
  x ≠ MHole ⇒ (⋀v. x = MVar v ⇒ P) ⇒ (⋀f Cs. x = MFun f Cs ⇒ P)
  ⇒ P
  by (cases x) auto

lemma prefix-comp-mctxt:
  C ≤ E ⇒ D ≤ E ⇒ (C, D) ∈ comp-mctxt
proof (induct E arbitrary: C D)
  case (MFun f Es C D)
  then show ?case
  proof (elim less-eq-mctxtE2)
    fix Cs Ds
    assume C: C = MFun f Cs and D: D = MFun f Ds
    and lC: length Cs = length Es and lD: length Ds = length Es
    and Ci: ⋀i. i < length Cs ⇒ Cs ! i ≤ Es ! i and Di: ⋀i. i < length Ds
    ⇒ Ds ! i ≤ Es ! i
    and IH: ⋀E' C' D'. E' ∈ set Es ⇒ C' ≤ E' ⇒ D' ≤ E' ⇒ (C', D') ∈
    comp-mctxt
    show (C, D) ∈ comp-mctxt
    by (auto simp: C D lC lD intro!: comp-mctxt.intros IH[OF - Ci Di])
  qed (auto intro: comp-mctxt.intros)
qed (auto elim: less-eq-mctxtE2(1,2) intro: comp-mctxt.intros)

lemma less-eq-mctxt-sup-conv1:
  (C, D) ∈ comp-mctxt ⇒ C ≤ D ⇔ C ⊔ D = D
  by (induct C D rule: comp-mctxt.induct) (auto elim!: less-eq-mctxtE2 nth-equalityE
  intro: nth-equalityI less-eq-mctxtI2(3))

lemma less-eq-mctxt-sup-conv2:
  (C, D) ∈ comp-mctxt ⇒ D ≤ C ⇔ C ⊔ D = C
  using less-eq-mctxt-sup-conv1[OF comp-mctxt-sym] by (auto simp: ac-simps)

lemma comp-mctxt-mctxt-of-term1[dest!]:
  (C, mctxt-of-term t) ∈ comp-mctxt ⇒ C ≤ mctxt-of-term t
proof (induct C mctxt-of-term t arbitrary: t rule: comp-mctxt.induct)
  case (MHole2 C t)
  then show ?case by (cases t, auto)
next

```

case ($MFun\ f\ g\ Cs\ Ds$)
then show $?case\ by\ (cases\ t,\ auto\ intro:\ less\text{-}eq\text{-}mctxtI2)$
qed *auto*

lemmas $comp\text{-}mctxt\text{-}mctxt\text{-}of\text{-}term2[dest!] = comp\text{-}mctxt\text{-}mctxt\text{-}of\text{-}term1[OF\ comp\text{-}mctxt\text{-}sym]$

lemma $mfun\text{-}leq\text{-}mfunI$:
 $f = g \implies length\ Cs = length\ Ds \implies (\bigwedge i. i < length\ Ds \implies Cs\ !\ i \leq Ds\ !\ i)$
 $\implies MFun\ f\ Cs \leq MFun\ g\ Ds$
by (*auto simp: less-eq-mctxt-def list-eq-iff-nth-eq*)

lemma $prefix\text{-}mctxt\text{-}sup$:
assumes $C \leq (E :: ('f, 'v)\ mctxt)\ D \leq E$ **shows** $C \sqcup D \leq E$
using *assms*
by (*induct E arbitrary: C D*) (*auto elim!: less-eq-mctxtE2 intro!: mfun-leq-mfunI*)

lemma $mreplace\text{-}at\text{-}leqI$:
 $p \in all\text{-}poss\text{-}mctxt\ C \implies C \leq E \implies D \leq subm\text{-}at\ E\ p \implies mreplace\text{-}at\ C\ p\ D \leq E$
by (*induct C p D arbitrary: E rule: mreplace-at.induct*)
(auto elim!: less-eq-mctxtE1 intro!: less-eq-mctxtI1 simp: upd-conv-take-nth-drop[symmetric] nth-list-update)

lemma $prefix\text{-}and\text{-}fewer\text{-}holes\text{-}implies\text{-}equal\text{-}mctxt$:
 $C \leq D \implies hole\text{-}poss\ C \subseteq hole\text{-}poss\ D \implies C = D$
proof (*induct C D rule: less-eq-mctxt-induct*)
case ($1\ D$) **then show** $?case\ by\ (cases\ D)\ auto$
next
case ($3\ Cs\ Ds\ f$)
have $i < length\ Ds \implies hole\text{-}poss\ (Cs\ !\ i) \subseteq hole\text{-}poss\ (Ds\ !\ i)$ **for** i **using** $3(1,4)$
by *auto*
then show $?case\ using\ 3\ by\ (auto\ intro!: nth\text{-}equalityI)$
qed *auto*

lemma $compare\text{-}mreplace\text{-}at$:
 $p \in poss\text{-}mctxt\ C \implies mreplace\text{-}at\ C\ p\ D \leq mreplace\text{-}at\ C\ p\ E \longleftrightarrow D \leq E$
proof (*induct C arbitrary: p*)
case ($MFun\ f\ Cs\ p$)
then show $?case$
by (*cases p, auto elim!: less-eq-mctxtE2(3) intro!: less-eq-mctxt-intros(3) simp: nth-append nth-Cons'*)
split: if-splits *auto*
qed *auto*

lemma $merge\text{-}mreplace\text{-}at$:
 $p \in poss\text{-}mctxt\ C \implies mreplace\text{-}at\ C\ p\ (D \sqcup E) = mreplace\text{-}at\ C\ p\ D \sqcup mreplace\text{-}at\ C\ p\ E$
proof (*induct C arbitrary: p*)

case ($MFun\ f\ Cs\ p$)
then show $?case$ **by** ($cases\ p$, $auto\ intro$: $nth-equalityI$)
qed $auto$

lemma *compare-mreplace-atI'*:

$C \leq D \implies C' \leq D' \implies p \in all-poss-mctxt\ C \implies mreplace-at\ C\ p\ C' \leq$
 $mreplace-at\ D\ p\ D'$

proof ($induct\ C\ D$ *arbitrary*: p *rule*: *less-eq-mctxt-induct*)

case ($\exists\ cs\ ds\ f\ p$)

then show $?case$ **by** ($cases\ p$, $auto\ intro!$: *less-eq-mctxt-intros*(\exists) *simp*: *nth-append*
nth-Cons')

qed $auto$

lemma *compare-mreplace-atI*:

$C \leq D \implies C' \leq D' \implies p \in poss-mctxt\ C \implies mreplace-at\ C\ p\ C' \leq mreplace-at$
 $D\ p\ D'$

using *compare-mreplace-atI'* *all-poss-mctxt-conv* **by** *blast*

lemma *all-poss-mctxt-mono*:

$C \leq D \implies all-poss-mctxt\ C \subseteq all-poss-mctxt\ D$

by ($induct\ C\ D$ *rule*: *less-eq-mctxt-induct*) *force+*

lemma *all-poss-mctxt-inf-mctxt*:

$(C, D) \in comp-mctxt \implies all-poss-mctxt\ (C \sqcap D) = all-poss-mctxt\ C \cap all-poss-mctxt$
 D

by ($induct\ C\ D$ *rule*: *comp-mctxt.induct*) $auto$

lemma *less-eq-subm-at*:

$p \in all-poss-mctxt\ C \implies C \leq D \implies subm-at\ C\ p \leq subm-at\ D\ p$

by ($induct\ C$ *arbitrary*: $p\ D$) ($auto\ elim$: *less-eq-mctxtE1*)

lemma *inf-subm-at*:

$p \in all-poss-mctxt\ (C \sqcap D) \implies subm-at\ (C \sqcap D)\ p = subm-at\ C\ p \sqcap subm-at$
 $D\ p$

proof ($induct\ C\ D$ *arbitrary*: p *rule*: *inf-mctxt.induct*)

case ($\lambda\ f\ Cs\ g\ Ds\ p$) **show** $?case$ **using** $\lambda(1)\ \lambda(2)$

by ($auto\ \lambda\ \lambda\ intro!$: $\lambda(1)[of\ (Cs\ !\ i,\ Ds\ !\ i)\ Cs\ !\ i\ Ds\ !\ i\ for\ i]$ *simp*: *set-zip*)

qed $auto$

lemma *less-eq-fill-holesI*:

assumes $length\ Ds = num-holes\ C\ length\ Es = num-holes\ C$

$\bigwedge i. i < num-holes\ C \implies Ds\ !\ i \leq Es\ !\ i$

shows $fill-holes-mctxt\ C\ Ds \leq fill-holes-mctxt\ C\ Es$

using *assms*($1,2$)[*symmetric*] *assms*(\exists)

by ($induct\ C\ Ds\ Es$ *rule*: *fill-holes-induct2*) ($auto\ intro!$: *less-eq-mctxtI1* *simp*:
partition-by-nth-nth)

lemma *less-eq-fill-holesD*:

```

assumes length Ds = num-holes C length Es = num-holes C
  fill-holes-mctxt C Ds ≤ fill-holes-mctxt C Es i < num-holes C
shows Ds ! i ≤ Es ! i
using assms(1,2)[symmetric] assms(3,4)
proof (induct C Ds Es arbitrary: i rule: fill-holes-induct2)
  case (MFun f Cs Ds Es)
  obtain j k where j < length Cs k < num-holes (Cs ! j)
    zip Ds Es ! i = partition-holes (zip Ds Es) Cs ! j ! k
  using nth-concat-split[of i partition-holes (zip Ds Es) Cs] MFun(1,2,5) by auto
  moreover then have f (zip Ds Es ! i) = partition-holes (map f (zip Ds Es))
    Cs ! j ! k for f
  using nth-map[of k partition-holes (zip Ds Es) Cs ! j f] MFun(1,2)
    length-partition-by-nth[of map num-holes Cs zip Ds Es] by simp
  from this[of fst] this[of snd] map-fst-zip[of Ds Es] map-snd-zip[of Ds Es]
  have Ds ! i = partition-holes Ds Cs ! j ! k Es ! i = partition-holes Es Cs ! j ! k
  using MFun(1,2,5) by simp-all
  ultimately show ?case using MFun(3)[of j k] MFun(1,2,4) by (auto elim:
less-eq-mctxtE1)
qed auto

```

```

lemma less-eq-fill-holes-iff:
  assumes length Ds = num-holes C length Es = num-holes C
  shows fill-holes-mctxt C Ds ≤ fill-holes-mctxt C Es  $\longleftrightarrow$  ( $\forall i < \text{num-holes } C. Ds$ 
! i ≤ Es ! i)
  using assms by (auto intro: less-eq-fill-holesI dest: less-eq-fill-holesD)

```

```

lemma fill-holes-mctxt-suffix[simp]:
  assumes length Ds = num-holes C shows C ≤ fill-holes-mctxt C Ds
  using assms(1)[symmetric]
  by (induct C Ds rule: fill-holes-induct) (auto simp: less-eq-mctxt-def intro: nth-equalityI)

```

```

lemma fill-holes-mctxt-id:
  assumes length Ds = num-holes C C = fill-holes-mctxt C Ds shows set Ds ⊆
{MHole}
  using assms(1)[symmetric] assms(2)
  apply (induct C Ds rule: fill-holes-induct)
  unfolding set-concat
  by (auto simp: set-conv-nth[of partition-holes - -] list-eq-iff-nth-eq[of - map - -])

```

```

lemma fill-holes-suffix[simp]:
  num-holes C = length ts  $\implies$  C ≤ mctxt-of-term (fill-holes C ts)
  by (induct C ts rule: fill-holes-induct) (auto intro: less-eq-mctxtI1)

```

7.6 An inverse of fill-holes

```

fun unfill-holes :: ('f, 'v) mctxt  $\Rightarrow$  ('f, 'v) term  $\Rightarrow$  ('f, 'v) 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  $\wedge$  length ts = length Cs then

```

```

concat (map (λi. unfill-holes (Cs ! i) (ts ! i)) [0..<length ts]) else undefined)

lemma length-unfill-holes[simp]:
  assumes C ≤ mctxt-of-term t
  shows length (unfill-holes C t) = num-holes C
  using assms
proof (induct C t rule: unfill-holes.induct)
  case (3 f Cs g ts) with 3(1)[OF - nth-mem] 3(2) show ?case
  by (auto simp: less-eq-mctxt-def length-concat
      intro!: cong[of sum-list, OF refl] nth-equalityI elim!: nth-equalityE)
qed (auto simp: less-eq-mctxt-def)

lemma fill-unfill-holes:
  assumes C ≤ mctxt-of-term t
  shows fill-holes C (unfill-holes C t) = t
  using assms
proof (induct C t rule: unfill-holes.induct)
  case (3 f Cs g ts) with 3(1)[OF - nth-mem] 3(2) show ?case
  by (auto simp: less-eq-mctxt-def intro!: fill-holes-arbitrary elim!: nth-equalityE)
qed (auto simp: less-eq-mctxt-def split: if-splits)

lemma unfill-fill-holes:
  assumes length ts = num-holes C
  shows unfill-holes C (fill-holes C ts) = ts
  using assms[symmetric]
proof (induct C ts rule: fill-holes-induct)
  case (MFun f Cs ts) then show ?case
  by (auto intro!: arg-cong[of - - concat] nth-equalityI[of - partition-holes ts Cs]
      simp del: concat-partition-by) auto
qed auto

lemma unfill-holes-subt:
  assumes C ≤ mctxt-of-term t and t' ∈ set (unfill-holes C t)
  shows t' ≤ t
  using assms
proof (induct C t rule: unfill-holes.induct)
  case (3 f Cs g ts)
  obtain i where i < length Cs and t' ∈ set (unfill-holes (Cs ! i) (ts ! i))
  using 3 by (auto dest!: in-set-idx split: if-splits simp: less-eq-mctxt-def)
  then show ?case
  using 3(1)[OF - nth-mem[of i]] 3(2,3) supseq.subt[of ts ! i ts t' g]
  by (auto simp: less-eq-mctxt-def elim!: nth-equalityE split: if-splits)
qed (auto simp: less-eq-mctxt-def split: if-splits)

lemma factor-hole-pos-by-prefix:
  assumes C ≤ D p ∈ hole-poss D
  obtains q where q ≤p p q ∈ hole-poss C
  using assms
  by (induct C D arbitrary: p thesis rule: less-eq-mctxt-induct)

```

(*auto*, *metis less-eq-pos-simps*(4))

lemma *concat-map-zip-upt*: **assumes** $\bigwedge i. i < n \implies \text{length } (f\ i) = \text{length } (g\ i)$
shows $\text{concat } (\text{map } (\lambda i. \text{zip } (f\ i) (g\ i))\ [0..<n]) = \text{zip } (\text{concat } (\text{map } f\ [0..<n]))$
 $(\text{concat } (\text{map } g\ [0..<n]))$
using *assms* **by** (*induct n arbitrary: f g*) (*auto simp: map-upt-Suc simp del: upt.simps*)

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 $\text{unfill-holes } (\text{fill-holes-mctxt } C\ Ds)\ t = \text{concat } (\text{map } (\lambda(D, t). \text{unfill-holes } D\ t)\ (\text{zip } Ds\ (\text{unfill-holes } C\ t)))$
using *assms*
proof (*induct C Ds arbitrary: t rule: fill-holes-induct*)
case (*MVar v*) **then show** ?*case* **by** (*cases t*) (*auto elim: less-eq-mctxtE1*)
next
case (*MFun f Cs Ds*)
have [*simp*]: $\text{length } ts = \text{length } Cs \implies \text{map } (\lambda i. \text{unfill-holes } (\text{map } (\lambda i. \text{fill-holes-mctxt } (Cs\ !\ i)\ (\text{partition-holes } Ds\ Cs\ !\ i))\ [0..<\text{length } Cs]\ !\ i)\ (ts\ !\ i))\ [0..<\text{length } Cs]$
 $= \text{map } (\lambda i. \text{unfill-holes } (\text{fill-holes-mctxt } (Cs\ !\ i)\ (\text{partition-holes } Ds\ Cs\ !\ i))\ (ts\ !\ i))\ [0..<\text{length } Cs]$ **for** *ts*
by (*auto intro: nth-equalityI*)
obtain *ts* **where** $\text{length } ts = \text{length } Cs\ t = \text{Fun } f\ ts$ **and**
 $\text{pre: } i < \text{length } Cs \implies \text{fill-holes-mctxt } (Cs\ !\ i)\ (\text{partition-holes } Ds\ Cs\ !\ i) \leq \text{mctxt-of-term } (ts\ !\ i)$ **for** *i*
using *MFun*(1,3) **by** (*cases t*) (*auto elim!: less-eq-mctxtE2*)
have *: $i \in \text{set } [0..<n] \implies i < n$ **for** *i n* **by** *auto*
have ***: $i < \text{length } Cs \implies Cs\ !\ i \leq \text{mctxt-of-term } (ts\ !\ i)$ **for** *i*
using *fill-holes-mctxt-suffix*[*of partition-holes Ds Cs ! i Cs ! i, OF length-partition-holes-nth*]
MFun(1) *pre*[*of i*]
by (*auto simp del: fill-holes-mctxt-suffix*)
have [*simp*]: $\text{concat } (\text{map } (\lambda i. \text{concat } (\text{map } f\ (\text{zip } (\text{partition-holes } Ds\ Cs\ !\ i)\ (\text{unfill-holes } (Cs\ !\ i)\ (ts\ !\ i)))))\ [0..<\text{length } Cs]) = \text{concat } (\text{map } f\ (\text{zip } Ds\ (\text{concat } (\text{map } (\lambda i. \text{unfill-holes } (Cs\ !\ i)\ (ts\ !\ i))\ [0..<\text{length } Cs])))$
for *f*
unfolding *concat-map-concat*[*of map - -, unfolded map-map comp-def*]
unfolding *map-map*[*of map f λi. zip (- i) (- i), symmetric, unfolded comp-def*]
map-concat[*symmetric*]
using *MFun*(1) *map-nth*[*of partition-holes Ds Cs*] **by** (*auto simp: length-unfill-holes*[*OF ****]
concat-map-zip-upt)
from *ts pre* **show** ?*case* **using** *MFun*(1) *map-cong*[*OF refl MFun*(2)[*OF **], *of*
 $[0..<\text{length } Cs]$ *id* $\lambda i. ts\ !\ i]$
by (*auto simp del: map-eq-conv*)
qed *auto*

lemma *unfill-holes-var-subst*:
 $C \leq \text{mctxt-of-term } t \implies \text{unfill-holes } (\text{map-vars-mctxt } f\ C)\ (t \cdot (\text{Var} \circ f)) = \text{map}$

```

( $\lambda t. t \cdot (\text{Var} \circ f)$ ) (unfill-holes  $C$   $t$ )
  by (induct  $C$   $t$  rule: unfill-holes.induct; (simp only: mctxt-of-term.simps; elim
less-eq-mctxtE2)?)
    (auto simp: map-concat intro!: arg-cong[of - - concat])

```

7.7 Ditto for fill-holes-mctxt

```

fun unfill-holes-mctxt :: ('f, 'v) mctxt  $\Rightarrow$  ('f, 'v) mctxt  $\Rightarrow$  ('f, 'v) mctxt list where
  unfill-holes-mctxt MHole  $D = [D]$ 
| unfill-holes-mctxt (MVar  $w$ ) (MVar  $v$ ) = (if  $v = w$  then [] else undefined)
| unfill-holes-mctxt (MFun  $g$   $Cs$ ) (MFun  $f$   $Ds$ ) = (if  $f = g \wedge \text{length } Ds = \text{length } Cs$ 
then
  concat (map ( $\lambda i. \text{unfill-holes-mctxt } (Cs ! i) (Ds ! i)$ ) [0.. $\text{length } Ds$ ]) else
undefined)

```

```

lemma length-unfill-holes-mctxt [simp]:
  assumes  $C \leq D$ 
  shows length (unfill-holes-mctxt  $C$   $D$ ) = num-holes  $C$ 
  using assms
proof (induct  $C$   $D$  rule: unfill-holes-mctxt.induct)
  case ( $\exists f$   $Cs$   $g$   $Ds$ ) with  $\exists(1)[OF - \text{nth-mem}] \exists(2)$  show ?case
    by (auto simp: less-eq-mctxt-def length-concat intro!: cong[of sum-list, OF refl]
nth-equalityI elim!: nth-equalityE)
qed (auto simp: less-eq-mctxt-def)

```

```

lemma fill-unfill-holes-mctxt:
  assumes  $C \leq D$ 
  shows fill-holes-mctxt  $C$  (unfill-holes-mctxt  $C$   $D$ ) =  $D$ 
  using assms
proof (induct  $C$   $D$  rule: unfill-holes-mctxt.induct)
  case ( $\exists f$   $Cs$   $g$   $Ds$ ) with  $\exists(1)[OF - \text{nth-mem}] \exists(2)$  show ?case
    by (auto simp: less-eq-mctxt-def intro!: fill-holes-arbitrary elim!: nth-equalityE)
qed (auto simp: less-eq-mctxt-def split: if-splits)

```

```

lemma unfill-fill-holes-mctxt:
  assumes length  $Ds = \text{num-holes } C$ 
  shows unfill-holes-mctxt  $C$  (fill-holes-mctxt  $C$   $Ds$ ) =  $Ds$ 
  using assms[symmetric]
proof (induct  $C$   $Ds$  rule: fill-holes-induct)
  case (MFun  $f$   $Cs$   $ts$ ) then show ?case
    by (auto intro!: arg-cong[of - - concat] nth-equalityI[of - partition-holes  $ts$   $Cs$ ]
simp del: concat-partition-by) auto
qed auto

```

```

lemma unfill-holes-mctxt-mctxt-of-term:
  assumes  $C \leq \text{mctxt-of-term } t$ 
  shows unfill-holes-mctxt  $C$  (mctxt-of-term  $t$ ) = map mctxt-of-term (unfill-holes
 $C$   $t$ )
  using assms

```

```

proof (induct C arbitrary: t)
  case (MVar x) then show ?case by (cases t) (auto elim: less-eq-mctxtE1)
next
  case MHole then show ?case by (cases t) (auto elim: less-eq-mctxtE1)
next
  case (MFun x1a x2) then show ?case
    by (cases t) (auto elim: less-eq-mctxtE1 simp: map-concat intro!: arg-cong[of -
- concat])
qed

```

7.8 Function symbols of prefixes

```

lemma funas-prefix[simp]:
   $C \leq D \implies fn \in \text{funas-mctxt } C \implies fn \in \text{funas-mctxt } D$ 
  unfolding less-eq-mctxt-def
proof (induct C D rule: inf-mctxt.induct)
  case (4 f Cs g Ds)
    from 4(3) obtain i where  $i < \text{length } Cs \wedge fn \in \text{funas-mctxt } (Cs ! i) \vee fn =$ 
    (f, length Cs)
    by (auto dest!: in-set-idx)
    moreover {
      assume  $i < \text{length } Cs \wedge fn \in \text{funas-mctxt } (Cs ! i)$ 
      then have  $i < \text{length } Ds \wedge fn \in \text{funas-mctxt } (Ds ! i)$  using 4(2)
      by (auto intro!: 4(1)[of - Cs ! i Ds ! i] split: if-splits elim!: nth-equalityE simp:
in-set-conv-nth)
      then have ?case by (auto)
    }
    ultimately show ?case using 4(2) by auto
qed auto

```

7.9 Parallel Rewriting using Multihole Contexts

```

datatype ('f,'v)par-info = Par-Info
  (par-left: ('f,'v)term)
  (par-right: ('f,'v)term)
  (par-rule: ('f,'v)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** $\text{par-rules} \equiv (\lambda \text{ info. par-rule ' set info})$

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** $\text{par-conds } R \equiv \lambda \text{ infos. Ball (set infos) (par-cond } R)$

lemma par-cond-imp-rrstep: **assumes** $\text{par-cond } R \text{ info}$
shows $(\text{par-left info}, \text{par-right info}) \in \text{rrstep } R$

```

using assms unfolding par-cond-def
by (metis rrstepE rrstepI singletonD)

lemma par-conds-imp-rrstep: assumes par-conds R infos
  and s = par-lefts infos ! i t = par-rights infos ! i
  and i < length infos
shows (s, t) ∈ rrstep R
proof -
  from assms have eq: s = par-left (infos ! i) t = par-right (infos ! i) and pc:
par-cond R (infos ! i)
  by auto
  show ?thesis unfolding eq using par-cond-imp-rrstep[OF pc] .
qed

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) ∈ par-rstep R
by (meson assms contra-subsetD par-rstep-mctxt rrstep-imp-rstep rstep-par-rstep)

definition par-rstep-mctxt where
  par-rstep-mctxt R C infos = {(s, t). s =f (C, par-lefts infos) ∧ t =f (C, par-rights
infos) ∧ par-conds R infos}

lemma par-rstep-mctxtI: assumes s =f (C, par-lefts infos) t =f (C, par-rights
infos) par-conds R infos
shows (s,t) ∈ par-rstep-mctxt R C infos
unfolding par-rstep-mctxt-def using assms by auto

lemma par-rstep-mctxt-reflI: (s,s) ∈ par-rstep-mctxt R (mctxt-of-term s) []
by (intro par-rstep-mctxtI, auto)

lemma par-rstep-mctxt-varI: (Var x, Var x) ∈ par-rstep-mctxt R (MVar x) []
by (intro par-rstep-mctxtI, auto)

lemma par-rstep-mctxt-MHoleI: (l,r) ∈ R ⇒ s = l · σ ⇒ t = r · σ ⇒ infos
= [Par-Info s t (l,r)]
 $\Rightarrow (s,t) \in \text{par-rstep-mctxt } R \text{ MHole infos}$ 
by (intro par-rstep-mctxtI, auto simp: par-cond-def)

lemma par-rstep-mctxt-funI:
  assumes rec:  $\bigwedge i. i < \text{length } ts \Rightarrow (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R (Cs ! i)$ 
(infos ! i)
  and len: length ss = length ts length Cs = length ts length infos = length ts
shows (Fun f ss, Fun f ts) ∈ par-rstep-mctxt R (MFun f Cs) (concat infos)
unfolding par-rstep-mctxt-def
proof (standard, unfold split, intro conjI)
  {
    fix i

```

```

    assume  $i < \text{length } ts$ 
    from  $\text{rec}[OF \text{ this, unfolded par-rstep-mctxt-def}]$ 
    have  $ss ! i =_f (Cs ! i, \text{par-lefts } (infos ! i)) \text{ } ts ! i =_f (Cs ! i, \text{par-rights } (infos !$ 
 $i))$ 
       $\text{par-conds } R (infos ! i) \text{ by auto}$ 
    } note  $* = \text{this}$ 
    from  $*(3)[\text{folded len}(3)]$  show  $\text{par-conds } R (\text{concat } infos)$ 
      by  $(\text{metis in-set-conv-nth nth-concat-split})$ 
    show  $\text{Fun } f \text{ } ss =_f (M\text{Fun } f \text{ } Cs, \text{par-lefts } (\text{concat } infos))$  unfolding  $\text{map-concat}$ 
      by  $(\text{intro eqf-MFunI, insert } *(1) \text{ len, auto})$ 
    show  $\text{Fun } f \text{ } ts =_f (M\text{Fun } f \text{ } Cs, \text{par-rights } (\text{concat } infos))$  unfolding  $\text{map-concat}$ 
      by  $(\text{intro eqf-MFunI, insert } *(2) \text{ len, auto})$ 
qed

```

lemma *par-rstep-mctxt-funI-ex*:

```

  assumes  $\bigwedge i. i < \text{length } ts \implies \exists C \text{ infos. } (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R \text{ } C$ 
 $infos$ 
    and  $\text{length } ss = \text{length } ts$ 
  shows  $\exists C \text{ infos. } (\text{Fun } f \text{ } ss, \text{Fun } f \text{ } ts) \in \text{par-rstep-mctxt } R \text{ } C \text{ infos} \wedge C \neq M\text{Hole}$ 

```

proof –

```

  let  $?n = \text{length } ts$ 
  from  $\text{assms}(1)$  have  $\forall i. \exists C \text{ infos. } i < ?n \longrightarrow (ss ! i, ts ! i) \in \text{par-rstep-mctxt}$ 
 $R \text{ } C \text{ infos}$  by auto
  from  $\text{choice}[OF \text{ this}]$  obtain  $C$  where  $\forall i. \exists \text{infos. } i < ?n \longrightarrow (ss ! i, ts ! i)$ 
 $\in \text{par-rstep-mctxt } R (C \text{ } i) \text{ infos}$  by auto
  from  $\text{choice}[OF \text{ this}]$  obtain  $infos$  where  $\text{steps: } \bigwedge i. i < ?n \implies (ss ! i, ts ! i)$ 
 $\in \text{par-rstep-mctxt } R (C \text{ } i) (\text{infos } i)$  by auto
  let  $?Cs = \text{map } C [0 ..< ?n]$ 
  let  $?Is = \text{map } infos [0 ..< ?n]$ 
  show  $?thesis$ 
  proof  $(\text{intro exI conjI, rule par-rstep-mctxt-funI})$ 
    show  $\text{length } ?Cs = ?n$  by simp
    show  $\text{length } ?Is = ?n$  by simp
  qed  $(\text{insert assms}(2) \text{ steps, auto})$ 
qed

```

lemma *par-rstep-mctxt-mono*: **assumes** $R \subseteq S$

```

  shows  $\text{par-rstep-mctxt } R \text{ } C \text{ infos} \subseteq \text{par-rstep-mctxt } S \text{ } C \text{ infos}$ 
  using  $\text{assms}$  unfolding par-rstep-mctxt-def par-cond-def by auto

```

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  $\text{par-conds } R \text{ } infos$ 
proof –
  have  $\exists C \text{ infos. } s =_f (C, \text{par-lefts } infos) \wedge t =_f (C, \text{par-rights } infos) \wedge \text{par-conds}$ 
 $R \text{ } infos$  (is  $\exists C \text{ infos. } ?P \text{ } s \text{ } t \text{ } C \text{ } infos)$ 

```

```

using assms
proof (induct)
  case (root-step s t  $\sigma$ )
    thus ?case by (intro exI[of - MHole] exI[of - [Par-Info (s ·  $\sigma$ ) (t ·  $\sigma$ ) (s,t)]],
auto simp: par-cond-def)
  next
    case (par-step-var x)
    show ?case by (intro exI[of - MVar x] exI[of - Nil], auto)
  next
    case (par-step-fun ts ss f)
    have  $\exists C \text{ infos. } (Fun\ f\ ss, Fun\ f\ ts) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos} \wedge C \neq MHole$ 
    by (intro par-rstep-mctxt-funI-ex, insert par-step-fun, auto simp: par-rstep-mctxt-def)
    then obtain C infos where  $(Fun\ f\ ss, Fun\ f\ ts) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos}$ 
by auto
    hence ?P (Fun f ss) (Fun f ts) C infos
    by (auto simp: par-rstep-mctxt-def)
    thus ?case by blast
  qed
with that show ?thesis by blast
qed

lemma par-rstep-par-rstep-mctxt-conv:
   $(s, t) \in par\text{-}rstep\ R \longleftrightarrow (\exists C \text{ infos. } (s, t) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos})$ 
proof
  assume  $(s, t) \in par\text{-}rstep\ R$ 
  from par-rstep-mctxtE[OF this] obtain C infos
  where  $s =_f (C, par\text{-}lefts\ \text{infos})$  and  $t =_f (C, par\text{-}rights\ \text{infos})$  and par-conds
R infos
  by metis
  then show  $\exists C \text{ infos. } (s, t) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos}$  by (auto simp: par-rstep-mctxt-def)
next
  assume  $\exists C \text{ infos. } (s, t) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos}$ 
  then show  $(s, t) \in par\text{-}rstep\ R$ 
  by (force simp: par-rstep-mctxt-def par-cond-def rstep-def' set-conv-nth intro!:
par-rstep-mctxt-rrstepI)
qed

fun subst-apply-par-info ::  $(f, v)par\text{-}info \Rightarrow (f, v)subst \Rightarrow (f, v)par\text{-}info$  (infixl
·pi 67) where
  Par-Info s t r ·pi  $\sigma = Par\text{-}Info\ (s \cdot \sigma)\ (t \cdot \sigma)\ r$ 

lemma subst-apply-par-info-simps[simp]:
  par-left (info ·pi  $\sigma$ ) = par-left info ·  $\sigma$ 
  par-right (info ·pi  $\sigma$ ) = par-right info ·  $\sigma$ 
  par-rule (info ·pi  $\sigma$ ) = par-rule info
  par-cond R info  $\Longrightarrow par\text{-}cond\ R\ (info \cdot pi\ \sigma)$ 
  unfolding par-cond-def
  by (cases info; force simp: subst.closedD subst-closed-rrstep) +

```

lemma *par-rstep-mctxt-subst*: **assumes** $(s, t) \in \text{par-rstep-mctxt } R \ C \ \text{infos}$
shows $(s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-mctxt } R \ (C \cdot \text{mc } \sigma) \ (\text{map } (\lambda i. i \cdot \pi \sigma) \ \text{infos})$
using *assms unfolding par-rstep-mctxt-def* **by** $(\text{auto simp: o-def dest!: subst-apply-mctxt-sound[of } - \ C \ - \ \sigma])$

lemma *par-rstep-mctxt-MVarE*:
assumes $(s, t) \in \text{par-rstep-mctxt } R \ (\text{MVar } x) \ \text{infos}$
shows $s = \text{Var } x \ t = \text{Var } x \ \text{infos} = []$
using *assms[unfolded par-rstep-mctxt-def]*
by $(\text{auto dest: eqf-MVarE})$

lemma *par-rstep-mctxt-MHoleE*:
assumes $(s, t) \in \text{par-rstep-mctxt } R \ \text{MHole} \ \text{infos}$
obtains *info* **where**
 $\text{par-left } \text{info} = s$
 $\text{par-right } \text{info} = t$
 $\text{infos} = [\text{info}]$
 $(s, t) \in \text{rrstep } R$
 $\text{par-cond } R \ \text{info}$
proof –
from *assms[unfolded par-rstep-mctxt-def, simplified]*
have $s =_f (\text{MHole}, \text{par-lefts } \text{infos}) \ t =_f (\text{MHole}, \text{par-rights } \text{infos})$ **and** *par-conds*
 $R \ \text{infos}$ **by** *auto*
from *eqf-MHoleE[OF this(1)] eqf-MHoleE[OF this(2)] this(3)*
obtain *info* **where** $*$: $\text{infos} = [\text{info}] \ s = \text{par-left } \text{info} \ t = \text{par-right } \text{info} \ \text{par-cond}$
 $R \ \text{info}$
by $(\text{cases } \text{infos}, \text{auto})$
from *par-cond-imp-rrstep[OF *(4)] **
have $(s, t) \in \text{rrstep } R$ **by** *auto*
with $*$ **have** $\exists \ \text{info}. \text{par-left } \text{info} = s \wedge \text{par-right } \text{info} = t \wedge \text{infos} = [\text{info}] \wedge (s,$
 $t) \in \text{rrstep } R \wedge$
 $\text{par-cond } R \ \text{info}$ **by** *auto*
thus $(\bigwedge \text{info}.$
 $\text{par-left } \text{info} = s \implies$
 $\text{par-right } \text{info} = t \implies$
 $\text{infos} = [\text{info}] \implies$
 $(s, t) \in \text{rrstep } R \implies \text{par-cond } R \ \text{info} \implies \text{thesis}) \implies$
 thesis **by** *blast*

qed

lemma *par-rstep-mctxt-MFunD*:
assumes $(s, t) \in \text{par-rstep-mctxt } R \ (\text{MFun } f \ Cs) \ \text{infos}$
shows $\exists \ ss \ ts \ \text{Infos}.$
 $s = \text{Fun } f \ ss \wedge$
 $t = \text{Fun } f \ ts \wedge$
 $\text{length } ss = \text{length } Cs \wedge$
 $\text{length } ts = \text{length } Cs \wedge$
 $\text{length } \text{Infos} = \text{length } Cs \wedge$
 $\text{infos} = \text{concat } \text{Infos} \wedge$

```

  (∀ i < length Cs. (ss ! i, ts ! i) ∈ par-rstep-mctxt R (Cs ! i) (Infos ! i))
proof -
  from assms[unfolded par-rstep-mctxt-def]
  have eq: s =f (MFun f Cs, par-lefts infos) t =f (MFun f Cs, par-rights infos)
    and pc: par-conds R infos
    by auto
  define Infos where Infos = partition-holes infos Cs
  let ?sss = map par-lefts Infos
  let ?tss = map par-rights Infos
  let ?n = length Cs
  let ?is = [0..n]
  from eqfE[OF eq(1)]
  have s: s = Fun f (map (λi. fill-holes (Cs ! i) (?sss ! i)) ?is)
    and num: num-holes (MFun f Cs) = length infos
    and len: length Infos = ?n
    and infos: infos = concat Infos
    and lens: ∧ i. i < ?n ⇒ num-holes (Cs ! i) = length (Infos ! i)
    by (auto simp: Infos-def)
  note pc = pc[unfolded infos set-concat]
  from eqfE[OF eq(2)] num
  have t: t = Fun f (map (λi. fill-holes (Cs ! i) (?tss ! i)) ?is)
    by (auto simp: Infos-def)
  show ?thesis
    apply (intro exI[of - Infos] exI conjI infos len allI impI)
      apply (rule s)
      apply (rule t)
      apply force
      apply force
      apply (intro par-rstep-mctxtI, insert lens len pc, auto)
    done
qed

```

end

8 Multi-Step Rewriting

```

theory Multistep
imports Trs
begin

```

Multi-step rewriting (without proof terms).

```

inductive-set
  mstep :: ('f, 'v) trs ⇒ ('f, 'v) term rel
  for R
where
  Var: (Var x, Var x) ∈ mstep R |
  args: ∧ f n ss ts. [length ss = n; length ts = n;

```

$\forall i < n. (ss ! i, ts ! i) \in mstep R \implies$
 $(Fun f ss, Fun f ts) \in mstep R \mid$
rule: $\bigwedge l r \sigma \tau. \llbracket (l, r) \in R; \forall x \in vars-term\ l. (\sigma x, \tau x) \in mstep R \rrbracket \implies$
 $(l \cdot \sigma, r \cdot \tau) \in mstep R$

lemma *mstep-refl [simp]*:

$(t, t) \in mstep R$

by (*induct t*) (*auto intro: mstep.intros*)

lemma *mstep-ctxt*:

assumes $(s, t) \in mstep R$

shows $(C\langle s \rangle, C\langle t \rangle) \in mstep R$

proof (*induction C*)

case *Hole* **with** *assms* **show** *?case* **by** *simp*

next

case (*More f ss C ts*)

let *?ss* = *ss @ C⟨s⟩ # ts*

let *?ts* = *ss @ C⟨t⟩ # ts*

{ fix i assume *i = length ss*

then have $(?ss ! i, ?ts ! i) \in mstep R$

using *More.IH* **by** *simp* **}**

moreover

{ fix i assume *i < length ss*

then have $(?ss ! i, ?ts ! i) \in mstep R$

by (*simp add: nth-append*) **}**

moreover

{ fix i assume *i < length ?ss and i > length ss*

then have $(?ss ! i, ?ts ! i) \in mstep R$

by (*simp add: nth-append*) **}**

ultimately

have $\forall i < length\ ?ss. (?ss ! i, ?ts ! i) \in mstep R$

by (*metis linorder-neqE-nat*)

from *mstep.args [OF - - this, simplified]*

show *?case* **by** *simp*

qed

lemma *rstep-imp-mstep*:

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

shows $(s, t) \in mstep R$

using *assms*

proof (*induct*)

case (*IH C σ l r*)

have $\forall x \in vars-term\ l. (\sigma x, \sigma x) \in mstep R$ **by** *simp*

from *mstep.rule [OF <(l, r) ∈ R> this]*

have $(l \cdot \sigma, r \cdot \sigma) \in mstep R$ **by** *simp*

from *mstep-ctxt [OF this]* **show** *?case* **by** *blast*

qed

lemma *rstep-mstep-subset*:

```

  rstep R ⊆ mstep R
  by (auto simp: rstep-imp-mstep)

lemma subst-rsteps-imp-rule-rsteps:
  assumes ∀ x ∈ vars-term l. (σ x, τ x) ∈ (rstep R)*
  and (l, r) ∈ R
  shows (l · σ, r · τ) ∈ (rstep R)*
proof -
  let ?σ = λ x. (if x ∈ vars-term l then σ x else τ x)
  have l · σ = l · ?σ
  by (simp add: term-subst-eq-conv)
  with ⟨l, r⟩ have (l · σ, r · ?σ) ∈ rstep R
  by auto
  moreover have (r · ?σ, r · τ) ∈ (rstep R)*
  by (rule subst-rsteps-imp-rsteps) (insert assms, auto)
  ultimately show ?thesis by auto
qed

lemma mstep-imp-rsteps:
  assumes (s, t) ∈ mstep R
  shows (s, t) ∈ (rstep R)*
using assms
proof (induct)
  case (args f n ss ts)
  then show ?case by (metis args-rsteps-imp-rsteps)
next
  case (rule l r σ τ)
  then show ?case using ⟨l, r⟩ by (metis subst-rsteps-imp-rule-rsteps)
qed simp

lemma mstep-rsteps-subset:
  shows mstep R ⊆ (rstep R)*
  by (auto simp: mstep-imp-rsteps)

lemma mstep-mono: R ⊆ S ⟹ mstep R ⊆ mstep S
proof -
  have (s, t) ∈ mstep R ⟹ R ⊆ S ⟹ (s, t) ∈ mstep S for s t
  by (induct rule: mstep.induct, auto intro: mstep.intros)
  thus R ⊆ S ⟹ mstep R ⊆ mstep S by auto
qed

Thus if mstep R has the diamond property, then rstep R is confluent.

lemma Var-mstep:
  assumes *: ⋀ l r. (l, r) ∈ R ⟹ ¬ is-Var l
  and (Var x, t) ∈ mstep R
  shows t = Var x
  using assms(2-)
proof cases
  case (rule l r σ τ)

```

```

    then show ?thesis using * by (cases l, auto)
qed auto

```

Maximal multi-step rewriting.

```

inductive-set
  mmstep :: ('f, 'v) trs  $\Rightarrow$  ('f, 'v) term rel
for R
where
  Var: (Var x, Var x)  $\in$  mmstep R |
  args:  $\bigwedge f n ss ts. \llbracket \text{length } ss = n; \text{length } ts = n; \neg (\exists (l, r) \in R. \exists \sigma. \text{Fun } f ss = l \cdot \sigma); \forall i < n. (ss ! i, ts ! i) \in \text{mmstep } R \rrbracket \Rightarrow$ 
    (Fun f ss, Fun f ts)  $\in$  mmstep R |
  rule:  $\bigwedge l r \sigma \tau. \llbracket (l, r) \in R; \forall x \in \text{vars-term } l. (\sigma x, \tau x) \in \text{mmstep } R \rrbracket \Rightarrow$ 
    (l  $\cdot$   $\sigma$ , r  $\cdot$   $\tau$ )  $\in$  mmstep R

```

```

lemma mmstep-imp-mstep:
  assumes (s, t)  $\in$  mmstep R
  shows (s, t)  $\in$  mstep R
  using assms by (induct) (auto intro: mstep.intros)

```

```

lemma mmstep-mstep-subset:
  mmstep R  $\subseteq$  mstep R
  by (auto simp: mmstep-imp-mstep)

```

end

```

theory Option-Util
imports Main
begin

```

```

primrec option-to-list :: 'a option  $\Rightarrow$  'a list
where
  option-to-list (Some a) = [a] |
  option-to-list None = []

```

```

lemma set-option-to-list-sound [simp]:
  set (option-to-list t) = set-option t
  by (induct t) auto

```

```

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*: $(s, t1) \in \text{par-rstep } R$ **and** *st2*: $(s, t2) \in \text{par-rstep } R$ **and** *weak-ortho*:

left-linear-trs $R \wedge b \ l \ r. (b, l, r) \in \text{critical-pairs ren } R \implies l = r$

and *wf*: $\bigwedge l \ r. (l, r) \in R \implies \text{is-Fun } l$

shows $\exists u. (t1, u) \in \text{par-rstep } R \wedge (t2, u) \in \text{par-rstep } R$

proof –

let *?R* = *par-rstep* *R*

let *?CP* = *critical-pairs ren* *R* *R*

{

fix *ls ts σ f r*

assume *below*: $\bigwedge i. i < \text{length } ls \implies ((\text{map } (\lambda l. l \cdot \sigma) \text{ } ls) ! i, ts ! i) \in ?R$

and *rule*: $(\text{Fun } f \text{ } ls, r) \in R$

and *len*: $\text{length } ts = \text{length } ls$

let *?ls* = $\text{map } (\lambda l. l \cdot \sigma) \text{ } ls$

from *weak-ortho*(1) **rule** *lin*: *linear-term* $(\text{Fun } f \text{ } ls)$ **unfolding** *left-linear-trs-def*

by *auto*

let *?p1* = $\lambda \tau i. ts ! i = ls ! i \cdot \tau \wedge (\forall x \in \text{vars-term } (ls ! i). (\sigma \ x, \tau \ x) \in \text{par-rstep } R)$

let *?p2* = $\lambda \tau i. (\exists C \ l'' \ l' \ r'. ls ! i = C \langle l'' \rangle \wedge \text{is-Fun } l'' \wedge (l', r') \in R \wedge (l'' \cdot \sigma = l' \cdot \tau) \wedge ((C \cdot_c \sigma) \langle r' \cdot \tau \rangle, ts ! i) \in \text{par-rstep } R)$

{

fix *i*

assume *i*: $i < \text{length } ls$

then **have** *i2*: $i < \text{length } ts$ **using** *len* **by** *simp*

from *below*[*OF* *i*] **have** *step*: $(ls ! i \cdot \sigma, ts ! i) \in ?R$ **using** *i* **by** *auto*

from *i* **have** *mem*: $ls ! i \in \text{set } ls$ **by** *auto*

from *lin* *i* **have** *lin*: *linear-term* $(ls ! i)$ **by** *auto*

from *par-rstep-linear-subst*[*OF* *lin* *step*] **have** $\exists \tau. ?p1 \ \tau \ i \vee ?p2 \ \tau \ i$.

} **note** *p12* = *this*

have $\exists u. (r \cdot \sigma, u) \in ?R \wedge (\text{Fun } f \text{ } ts, u) \in ?R$

proof (*cases* $\exists i \ \tau. i < \text{length } ls \wedge ?p2 \ \tau \ i$)

case *True*

then **obtain** *i τ* **where** *i*: $i < \text{length } ls$ **and** *p2*: $?p2 \ \tau \ i$ **by** *blast*

from *p2* **obtain** *C l'' l' r'* **where** *lsi*: $ls ! i = C \langle l'' \rangle$ **and** *l''*: *is-Fun* (l'')

and *lr'*: $(l', r') \in R$

and *unif*: $l'' \cdot \sigma = l' \cdot \tau$ **and** *tsi*: $((C \cdot_c \sigma) \langle r' \cdot \tau \rangle, ts ! i) \in ?R$ **by** *blast*

from *id-take-nth-drop*[*OF* *i*] **obtain** *bef aft* **where** *ls*: $ls = \text{bef} @ C \langle l'' \rangle \#$

aft **and** *bef*: *bef* = *take* *i* *ls* **unfolding** *lsi* **by** *auto*

from *i* *bef* **have** *bef*: $\text{length } bef = i$ **by** *auto*

let *?C* = *More* *f* *bef* *C* *aft*

from *bef* **have** *hp*: *hole-pos* *?C* = *i* # *hole-pos* *C* **by** *simp*

have *fls*: $\text{Fun } f \text{ } ls = ?C \langle l'' \rangle$ **unfolding** *ls* **by** *simp*

```

from mgu-vd-complete[OF unif] obtain  $\mu 1$   $\mu 2$   $\delta$  where
  mgu: mgu-vd ren  $l''$   $l' = \text{Some } (\mu 1, \mu 2)$  and id:  $l'' \cdot \mu 1 = l' \cdot \mu 2$ 
  and sigma:  $\sigma = \mu 1 \circ_s \delta$  and tau:  $\tau = \mu 2 \circ_s \delta$  by blast
let ?sig = map ( $\lambda s. s \cdot \sigma$ )
let ?r = ( $C \cdot_c \sigma$ )  $\langle r' \cdot \tau \rangle$ 
let ?bra = ?sig bef @ ?r # ?sig aft
from weak-ortho(2)[OF critical-pairsI[OF rule lr' fls l'' mgu refl refl refl]]

have id:  $r \cdot \sigma = (?C \cdot_c \sigma) \langle r' \cdot \tau \rangle$  unfolding sigma tau by simp
also have ... = Fun f ?bra by simp
also have (... , Fun f ts)  $\in ?R$ 
proof (rule par-rstep.par-step-fun)
  show length ?bra = length ts unfolding len unfolding ls by simp
next
  fix j
  assume j:  $j < \text{length } ts$ 
  show (?bra ! j, ts ! j)  $\in ?R$ 
  proof (cases j = i)
    case True
    then have ?bra ! j = ?r using bef i by (simp add: nth-append)
    then show ?thesis using tsi True by simp
  next
    case False
    then have ?bra ! j = (?sig bef @ (C  $\langle l'' \rangle \cdot \sigma$ ) # ?sig aft) ! j using False
  bef i by (simp add: nth-append)
  also have ... = ?sig ls ! j unfolding ls by simp
  finally show ?thesis
    using below[OF j[unfolded len]] by auto
  qed
qed
finally have step:  $(r \cdot \sigma, \text{Fun } f \text{ ts}) \in ?R$  .
show  $\exists u. (r \cdot \sigma, u) \in ?R \wedge (\text{Fun } f \text{ ts}, u) \in ?R$ 
  by (rule exI, rule conjI[OF step par-rstep-refl])
next
  case False
  with p12
  have  $\forall i. (\exists \tau. i < \text{length } ls \longrightarrow ?p1 \ \tau \ i)$  by blast
  from choice[OF this] obtain tau where tau:  $\bigwedge i. i < \text{length } ls \implies ?p1 \ (\text{tau } i) \ i$  by blast
  from lin have is-partition (map vars-term ls) by auto
  from subst-merge[OF this, of tau] obtain  $\tau$  where  $\tau: \bigwedge i \ x. i < \text{length } ls$ 
 $\implies x \in \text{vars-term } (ls ! i) \implies \tau \ x = \text{tau } i \ x$ 
  by blast
  obtain  $\delta$  where delta:  $\delta = (\lambda x. \text{if } x \in \text{vars-term } (\text{Fun } f \text{ ls}) \text{ then } \tau \ x \text{ else } \sigma$ 
  x) by auto
  {
    fix i
    assume i:  $i < \text{length } ls$ 
    from tau[OF i] have p: ?p1 (tau i) i .
  }

```

```

    have id1:  $ls ! i \cdot \tau i = ls ! i \cdot \tau$ 
      by (rule term-subst-eq[OF  $\tau$ [OF  $i$ , symmetric]])
    have id2:  $\dots = ls ! i \cdot \delta$ 
      by (rule term-subst-eq, unfold delta, insert  $i$ , auto)
    have p:  $?p1 \ \delta \ i$  using  $p$  using  $\tau$ [OF  $i$ ] unfolding id1 id2 using id2
  unfolding delta by auto
} note delt = this
have r-delt:  $(r \cdot \sigma, r \cdot \delta) \in ?R$ 
proof (rule all-ctxt-closed-subst-step)
  fix  $x$ 
  assume  $x: x \in \text{vars-term } r$ 
  show  $(\sigma x, \delta x) \in ?R$ 
  proof (cases  $x \in \text{vars-term } (Fun f ls)$ )
    case True
    then obtain  $l$  where  $l: l \in \text{set } ls$  and  $x: x \in \text{vars-term } l$  by auto
    from  $l$ [unfolded set-conv-nth] obtain  $i$  where  $i: i < \text{length } ls$  and  $l: l =$ 
 $ls ! i$  by auto
    from delt[OF  $i$ ]  $x \ l$  show ?thesis by auto
  next
  case False
  then have  $\delta x = \sigma x$  unfolding delta by auto
  then show ?thesis by auto
qed
qed auto
{
  let ?ls = map  $(\lambda l. l \cdot \delta)$   $ls$ 
  have  $ts = \text{map } (\lambda i. ts ! i) [0 ..< \text{length } ts]$  by (rule map-nth[symmetric])
  also have  $\dots = \text{map } (\lambda i. ts ! i) [0 ..< \text{length } ls]$  unfolding len by simp
  also have  $\dots = \text{map } (\lambda i. ?ls ! i) [0 ..< \text{length } ?ls]$ 
    by (rule nth-map-conv, insert delt[THEN conjunct1], auto)
  also have  $\dots = ?ls$ 
    by (rule map-nth)
  finally have  $Fun f ts = Fun f ls \cdot \delta$  by simp
} note id = this
have l-delt:  $(Fun f ts, r \cdot \delta) \in ?R$  unfolding id
  by (rule par-rstep.root-step[OF rule])
show  $\exists u. (r \cdot \sigma, u) \in ?R \wedge (Fun f ts, u) \in ?R$ 
  by (intro exI conjI, rule r-delt, rule l-delt)
qed
} note root-arg = this
from st1 st2 show ?thesis
proof (induct arbitrary: t2 rule: par-rstep.induct)
  case (par-step-var  $x \ t2$ )
  have  $t2: t2 = Var x$ 
    by (rule wf-trs-par-rstep[OF wf par-step-var])
  show  $\exists u. (Var x, u) \in ?R \wedge (t2, u) \in ?R$  unfolding t2
    by (intro conjI exI par-rstep.par-step-var, auto)
next
  case (par-step-fun ts1 ss f t2)

```

```

note IH = this
show ?case using IH(4)
proof (cases rule: par-rstep.cases)
  case (par-step-fun ts2)
  from IH(3) par-step-fun(3) have len: length ts2 = length ts1 by simp
  {
    fix i
    assume i: i < length ts1
    then have i2: i < length ts2 using len by simp
    from par-step-fun(2)[OF i2] have step2: (ss ! i, ts2 ! i) ∈ ?R .
    from IH(2)[OF i step2] have ∃ u. (ts1 ! i, u) ∈ ?R ∧ (ts2 ! i, u) ∈ ?R .
  }
  then have ∀ i. ∃ u. (i < length ts1 ⟶ (ts1 ! i, u) ∈ ?R ∧ (ts2 ! i, u) ∈
?R) by blast
  from choice[OF this] obtain us where join: ∧ i. i < length ts1 ⟹ (ts1 !
i, us i) ∈ ?R ∧ (ts2 ! i, us i) ∈ ?R by blast
  let ?us = map us [0 ..< length ts1]
  {
    fix i
    assume i: i < length ts1
    from join[OF this] i have (ts1 ! i, ?us ! i) ∈ ?R (ts2 ! i, ?us ! i) ∈ ?R by
auto
  } note join = this
  let ?u = Fun f ?us
  have step1: (Fun f ts1, ?u) ∈ ?R
    by (rule par-rstep.par-step-fun[OF join(1)], auto)
  have step2: (Fun f ts2, ?u) ∈ ?R
    by (rule par-rstep.par-step-fun[OF join(2)], insert len, auto)
  show ?thesis unfolding par-step-fun(1) using step1 step2 by blast
next
case (root-step l r σ)
from wf[OF root-step(3)] root-step(1) obtain ls where l: l = Fun f ls
  by auto
from root-step(1) l have ss: ss = map (λ l. l · σ) ls (is - = ?ls) by simp
from root-step(3) l have rule: (Fun f ls, r) ∈ R by simp
from root-step(2) have t2: t2 = r · σ .
from par-step-fun(3) ss have len: length ts1 = length ls by simp
from root-arg[OF par-step-fun(1)[unfolded ss len] rule len]
show ?thesis unfolding t2 by blast
qed
next
case (root-step l r σ)
note IH = this
from wf[OF IH(1)] IH(1) obtain f ls where l: l = Fun f ls and rule: (Fun f
ls, r) ∈ R
  by (cases l, auto)
from IH(2)[unfolded l] show ?case
proof (cases rule: par-rstep.cases)
  case (par-step-var x)

```

```

    then show ?thesis by simp
  next
    case (root-step l' r'  $\tau$ )
    then have t2:  $t2 = r' \cdot \tau$  by auto
    have id:  $\text{Fun } f \text{ } ls = \square \langle \text{Fun } f \text{ } ls \rangle$  by simp
    from mgu-vd-complete[OF root-step(1), of ren] obtain mu1 mu2 delta where
      mgu: mgu-vd ren ( $\text{Fun } f \text{ } ls$ )  $l' = \text{Some } (mu1, mu2)$  and sigma:  $\sigma = mu1 \circ_s$ 
      delta
    and tau:  $\tau = mu2 \circ_s \text{ delta}$  by auto
    from weak-ortho(2)[OF critical-pairsI[OF rule root-step(3) id - mgu refl refl]]
    have  $r \cdot mu1 = r' \cdot mu2$  by simp
    then have id:  $r \cdot \sigma = r' \cdot \tau$  unfolding sigma tau by simp
    show ?thesis unfolding t2 id by auto
  next
    case (par-step-fun ts ls' g)
    then have ls':  $ls' = \text{map } (\lambda l. l \cdot \sigma) \text{ } ls$  and g:  $g = f$  and len:  $\text{length } ts =$ 
      length ls by auto
    note par-step-fun = par-step-fun[unfolded ls' g len]
    from root-arg[OF par-step-fun(3) rule len]
    show ?thesis unfolding par-step-fun(2) .
  qed
qed
qed

```

lemma weakly-orthogonal-par-rstep-CR:
 assumes weak-ortho: $\text{left-linear-trs } R \wedge b \text{ } l \text{ } r. (b,l,r) \in \text{critical-pairs ren } R \text{ } R$
 $\implies l = r$
 and wf: $\bigwedge l \text{ } r. (l,r) \in R \implies \text{is-Fun } l$
 shows CR (par-rstep R)
 proof -
 let ?R = par-rstep R
 from weakly-orthogonal-main[OF - - weak-ortho wf]
 have diamond: $\bigwedge s \text{ } t1 \text{ } t2. (s,t1) \in ?R \implies (s,t2) \in ?R \implies \exists u. (t1,u) \in ?R \wedge$
 $(t2,u) \in ?R$.
 show ?thesis
 by (rule diamond-imp-CR, rule diamond-I, insert diamond, blast)
 qed

lemma weakly-orthogonal-rstep-CR:
 assumes weak-ortho: $\text{left-linear-trs } R \wedge b \text{ } l \text{ } r. (b,l,r) \in \text{critical-pairs ren } R \text{ } R$
 $\implies l = r$
 and wf: $\bigwedge l \text{ } r. (l,r) \in R \implies \text{is-Fun } l$
 shows CR (rstep R)
 proof -
 from weakly-orthogonal-par-rstep-CR[OF assms] have CR (par-rstep R) .
 then show ?thesis unfolding CR-on-def join-def rtrancl-converse par-rsteps-rsteps
 .

qed

end

end

theory *Par-Step-Var-Restricted*

imports

Trs

Multihole-Context

SubList

begin

fun *vars-below-hole* :: $(f, v)term \Rightarrow (f, v)mctxt \Rightarrow 'v \text{ set}$ **where**
 vars-below-hole *t* *MHole* = *vars-term* *t*
| *vars-below-hole* *t* (*MVar* *y*) = {}
| *vars-below-hole* (*Fun* - *ts*) (*MFun* - *Cs*) =
 $\bigcup (\text{set } (\text{map } (\lambda (t, C). \text{vars-below-hole } t \ C) (\text{zip } ts \ Cs)))$
| *vars-below-hole* (*Var* -) (*MFun* -) = *Code.abort* (*STR* "assumption in vars-below-hole
violated") ($\lambda -. \{\}$)

lemma *vars-below-hole-no-hole*: $\text{hole-poss } C = \{\} \implies \text{vars-below-hole } t \ C = \{\}$
by (*induct* *t* *C* *rule*: *vars-below-hole.induct*, *auto* *simp*: *set-zip*, *blast*)

lemma *vars-below-hole-mctxt-of-term*[*simp*]: $\text{vars-below-hole } t \ (\text{mctxt-of-term } u) = \{\}$
by (*rule* *vars-below-hole-no-hole*, *auto*)

lemma *vars-below-hole-vars-term*: $\text{vars-below-hole } t \ C \subseteq \text{vars-term } t$
by (*induct* *t* *C* *rule*: *vars-below-hole.induct*; *force* *simp*: *set-zip* *set-conv-nth*)

lemma *vars-below-hole-subst*[*simp*]: $\text{vars-below-hole } t \ (C \cdot mc \ \sigma) = \text{vars-below-hole } t \ C$
by (*induct* *t* *C* *rule*: *vars-below-hole.induct*; *fastforce* *simp*: *set-zip*)

lemma *vars-below-hole-Fun*: **assumes** $\text{length } ls = \text{length } Cs$
shows $\text{vars-below-hole } (\text{Fun } f \ ls) \ (\text{MFun } f \ Cs) = \bigcup \{\text{vars-below-hole } (ls \ ! \ i) \ (Cs \ ! \ i) \mid i. i < \text{length } Cs\}$
using *assms* **by** (*auto* *simp*: *set-zip*)

lemma *vars-below-hole-term-subst*:
 $\text{hole-poss } D \subseteq \text{poss } t \implies \text{vars-below-hole } (t \cdot \sigma) \ D = \bigcup (\text{vars-term } ' \ \sigma \ ' \ \text{vars-below-hole } t \ D)$
proof (*induct* *t* *D* *rule*: *vars-below-hole.induct*)

```

    case (1 t)
    then show ?case by (auto simp: vars-term-subst)
next
    case (3 f ts g Cs)
    then show ?case by (fastforce simp: set-zip)
next
    case (4 x f Cs)
    hence hp: hole-poss (MFun f Cs) = {} by auto
    show ?case unfolding vars-below-hole-no-hole[OF hp] by auto
qed auto

```

```

lemma vars-below-hole-egf: assumes  $t =_f (C, ts)$ 
  shows vars-below-hole  $t C = \bigcup (vars-term \text{ ' set } ts)$ 
  using assms
proof (induct C arbitrary: t ts)
  case (MVar x)
  from egf-MVarE[OF MVar(1)]
  show ?case by auto
next
  case MHole
  from egf-MHoleE[OF MHole(1)]
  show ?case by auto
next
  case (MFun f Cs t ss)
  from egf-MFunE[OF MFun(2)] obtain ts sss where
    *:  $t = Fun f ts$   $length\ ts = length\ Cs$   $length\ sss = length\ Cs$ 
     $\bigwedge i. i < length\ Cs \implies ts ! i =_f (Cs ! i, sss ! i)$ 
     $ss = concat\ sss$  by blast
  {
    fix i
    assume i:  $i < length\ Cs$ 
    hence mem:  $Cs ! i \in set\ Cs$  by auto
    from MFun(1)[OF mem *(4)[OF i]]
    have vars-below-hole  $(ts ! i) (Cs ! i) = \bigcup (vars-term \text{ ' set } (sss ! i))$  .
  } note IH = this
  show ?case unfolding *(1) *(5) set-concat set-conv-nth[of sss] using IH *(2,3)
    by (auto simp: set-zip)
qed

```

definition $par\text{-}rstep\text{-}var\text{-}restr\ R\ V = \{(s, t) \mid s\ t\ C\ \text{infos.}$
 $(s, t) \in par\text{-}rstep\text{-}mctxt\ R\ C\ \text{infos} \wedge vars\text{-}below\text{-}hole\ t\ C \cap V = \{\}\}$

lemma $par\text{-}rstep\text{-}var\text{-}restr\text{-}mono$: assumes $R \subseteq S\ W \subseteq V$
 shows $par\text{-}rstep\text{-}var\text{-}restr\ R\ V \subseteq par\text{-}rstep\text{-}var\text{-}restr\ S\ W$
 unfolding $par\text{-}rstep\text{-}var\text{-}restr\text{-}def$ using $par\text{-}rstep\text{-}mctxt\text{-}mono$ [OF assms(1)] assms(2)
 by blast

```

lemma par-rstep-var-restr-refl[simp]:  $(t, t) \in \text{par-rstep-var-restr } R \ V$ 
  unfolding par-rstep-var-restr-def
  by (intro CollectI exI conjI refl, force, rule par-rstep-mctxt-refl, auto)

```

```

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  $(s \cdot \delta, t \cdot \gamma) \in \text{par-rstep } R$ 
proof -
  from st[unfolded par-rstep-var-restr-def] subst-eq
  obtain C infos where st:  $(s, t) \in \text{par-rstep-mctxt } R \ C \ \text{infos}$ 
  and subst-eq:  $\bigwedge x. x \in \text{vars-below-hole } t \ C \implies \delta x = \gamma x$ 
  by auto
  thus ?thesis
proof (induct C arbitrary: s t infos)
  case (MVar x)
  from par-rstep-mctxt-MVarE[OF this(1)]
  show ?case using subst-R by auto
next
  case (MHole s t)
  have  $(s, t) \in \text{par-rstep } R$ 
  using MHole.prem(1) par-rstep-par-rstep-mctxt-conv by blast
  hence step:  $(s \cdot \delta, t \cdot \delta) \in \text{par-rstep } R$ 
  by (rule subst-closed-par-rstep)
  have  $\text{vars-below-hole } t \ \text{MHole} = \text{vars-term } t$  by simp
  with MHole(2) have t:  $t \cdot \delta = t \cdot \gamma$  by (auto intro: term-subst-eq)
  thus ?case using step by auto
next
  case (MFun f Cs s t infos)
  let ?n = length Cs
  let ?is = [0..n]
  from par-rstep-mctxt-MFunD[OF MFun(2)]
  obtain ss ts Infos
  where s:  $s = \text{Fun } f \ ss$ 
  and t:  $t = \text{Fun } f \ ts$ 
  and len:  $\text{length } ss = \text{length } Cs$ 
   $\text{length } ts = \text{length } Cs$ 
   $\text{length } \text{Infos} = \text{length } Cs$ 
  and infos:  $\text{infos} = \text{concat } \text{Infos}$ 
  and steps:  $\bigwedge i. i < \text{length } Cs \implies (ss ! i, ts ! i) \in \text{par-rstep-mctxt } R \ (Cs ! i)$ 
  (Infos ! i)
  by blast
  {
  fix i
  assume i:  $i < ?n$ 

```

```

    hence mem:  $Cs ! i \in \text{set } Cs$  by auto
    have IH:  $(ss ! i \cdot \delta, ts ! i \cdot \gamma) \in \text{par-rstep } R$ 
    proof (rule MFun(1)[OF mem steps[OF i]])
      fix x
      assume  $x \in \text{vars-below-hole } (ts ! i) (Cs ! i)$ 
      hence  $x \in \text{vars-below-hole } t (MFun f Cs)$  unfolding  $t$  using  $i \text{ len}(2)$ 
      by (auto simp: set-zip)
      from MFun(3)[OF this] show  $\delta x = \gamma x$  .
    qed
  }
  thus ?case unfolding  $s t$  using  $\text{len}(1-2)$  MFun(1-2) by auto
qed
qed

```

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  $(s, t) \in \text{par-rstep-var-restr } R (\gamma \cdot V)$ 
  and  $\bigwedge x. \sigma x \cdot (Var \circ \gamma) = Var x$ 
shows  $(s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-var-restr } R V$ 
proof -
  from assms(1)[unfolded par-rstep-var-restr-def, simplified]
  obtain  $C$  infos where  $\text{step}: (s, t) \in \text{par-rstep-mctxt } R C$  infos and  $\text{vars:}$ 
 $\text{vars-below-hole } t C \cap \gamma \cdot V = \{\}$ 
  by auto
  from step[unfolded par-rstep-mctxt-def, simplified]
  have  $t =_f (C, \text{par-rights infos})$  by auto
  hence  $\text{hole-poss } C \subseteq \text{poss } t$  by (metis hole-poss-subset-poss)
  hence  $hp: \text{hole-poss } (C \cdot mc \sigma) \subseteq \text{poss } t$ 
  using hole-poss-subst by auto
  from par-rstep-mctxt-subst[OF step, of  $\sigma$ ]
  have  $\text{step}: (s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-mctxt } R (C \cdot mc \sigma) (\text{map } (\lambda i. i \cdot pi \sigma) \text{ infos})$ 
  .
  show  $(s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep-var-restr } R V$ 
  unfolding par-rstep-var-restr-def
proof (standard, intro exI conjI, rule refl, rule step)
  show  $\text{vars-below-hole } (t \cdot \sigma) (C \cdot mc \sigma) \cap V = \{\}$ 
  unfolding vars-below-hole-term-subst[OF hp]
  unfolding vars-below-hole-subst
proof (intro equalsOI, elim IntE)
  fix x
  assume  $x \in \bigcup (\text{vars-term } \sigma \cdot \text{vars-below-hole } t C)$ 
  then obtain  $y$  where  $y: y \in \text{vars-below-hole } t C$  and  $x: x \in \text{vars-term } (\sigma$ 
 $y)$  by auto
  from  $y$  vars have  $y: y \notin \gamma \cdot V$  by auto
  assume  $x \in V$ 
  with assms(2)[of  $y$ ]  $y x$  show False unfolding o-def by (cases  $\sigma y$ , auto)
qed

```

```
qed
qed
```

```
end
```

9 Permutations

```
theory Renaming
imports
  HOL-Library.Adhoc-Overloading
  HOL-Library.Infinite-Set
  Fresh-Identifiers.Fresh
  Preliminaries
begin
```

This theory is mainly ported from HOL-Nominal2, but using locales instead of type classes. The intention is to leave the type of atoms arbitrary (such that it can later be used with polymorphic first-order terms).

The set of all permutations of a given type.

```
definition perms :: ('a  $\Rightarrow$  'a) set
where
  perms = {f. bij f  $\wedge$  finite {x. f x  $\neq$  x}}
```

```
typedef 'a perm = perms :: ('a  $\Rightarrow$  'a) set
by standard (auto simp: perms-def)
```

```
lemma permsI [Pure.intro]:
  assumes bij f and MOST x. f x = x
  shows f  $\in$  perms
  using assms by (auto simp: perms-def) (metis MOST-iff-finiteNeg)
```

```
lemma perms-imp-bij:
  f  $\in$  perms  $\implies$  bij f
  by (simp add: perms-def)
```

```
lemma perms-imp-finite-domain:
  f  $\in$  perms  $\implies$  finite {x. f x  $\neq$  x}
  by (simp add: perms-def)
```

```
lemma perms-imp-MOST-eq:
  f  $\in$  perms  $\implies$  MOST x. f x = x
  by (simp add: perms-def) (metis MOST-iff-finiteNeg)
```

```
lemma id-perms [simp]:
  id  $\in$  perms
  ( $\lambda x. x$ )  $\in$  perms
  by (auto simp: perms-def bij-def)
```

```

lemma perms-comp [simp]:
  assumes  $f \in \text{perms}$  and  $g \in \text{perms}$ 
  shows  $(f \circ g) \in \text{perms}$ 
  using assms
  by (force intro: permsI bij-comp elim: perms-imp-bij MOST-rev-mp [OF perms-imp-MOST-eq])

lemma perms-inv:
  assumes  $f \in \text{perms}$ 
  shows  $\text{inv } f \in \text{perms}$ 
  using assms
  by (force intro: permsI bij-imp-bij-inv MOST-mono [OF perms-imp-MOST-eq]
    dest: perms-imp-bij
    simp: bij-def inv-f-eq)

lemma bij-Rep-perm:
  bij (Rep-perm  $p$ )
  using Rep-perm [of p] by (simp add: perms-def)

lemma finite-Rep-perm:
  finite  $\{x. \text{Rep-perm } p \ x \neq x\}$ 
  using Rep-perm [of p] by (simp add: perms-def)

lemma Rep-perm-ext:
  Rep-perm  $p1 = \text{Rep-perm } p2 \implies p1 = p2$ 
  by (simp add: fun-eq-iff Rep-perm-inject [symmetric])

instance perm :: (type) size ..

instantiation perm :: (type) group-add
begin

definition  $0 = \text{Abs-perm } \text{id}$ 
definition  $- \ p = \text{Abs-perm } (\text{inv } (\text{Rep-perm } p))$ 
definition  $p + q = \text{Abs-perm } (\text{Rep-perm } p \circ \text{Rep-perm } q)$ 
definition  $(p1::'a \text{ perm}) - p2 = p1 + - \ p2$ 

lemma Rep-perm-0: Rep-perm  $0 = \text{id}$ 
  unfolding zero-perm-def by (simp add: Abs-perm-inverse)

lemma Rep-perm-add:
  Rep-perm  $(p1 + p2) = \text{Rep-perm } p1 \circ \text{Rep-perm } p2$ 
  unfolding plus-perm-def by (simp add: Abs-perm-inverse Rep-perm)

lemma Rep-perm-uminus:
  Rep-perm  $(- \ p) = \text{inv } (\text{Rep-perm } p)$ 
  unfolding uminus-perm-def by (simp add: Abs-perm-inverse perms-inv Rep-perm)

instance

```

by *standard*
 (*simp-all add: Rep-perm-inject [symmetric] minus-perm-def Rep-perm-add*
Rep-perm-uminus
Rep-perm-0 o-assoc inv-o-cancel [OF bij-is-inj [OF bij-Rep-perm]])

end

definition *swap* :: 'a \Rightarrow 'a \Rightarrow 'a *perm* ('(- \rightleftharpoons -'))

where

(*x* \rightleftharpoons *y*) = *Abs-perm* ($\lambda z.$ *if* *z* = *x* *then* *y* *else* *if* *z* = *y* *then* *x* *else* *z*)

lemma *Rep-perm-swap*:

Rep-perm (*x* \rightleftharpoons *y*) = ($\lambda z.$ *if* *z* = *x* *then* *y* *else* *if* *z* = *y* *then* *x* *else* *z*)

by (*auto intro!*; *Abs-perm-inverse permsI simp: bij-def MOST-eq-imp inj-on-def*
MOST-conj-distrib swap-def)

lemmas *Rep-perm-simps* =

Rep-perm-0
Rep-perm-add
Rep-perm-uminus
Rep-perm-swap

lemma *swap-cancel*:

(*x* \rightleftharpoons *y*) + (*x* \rightleftharpoons *y*) = 0

(*x* \rightleftharpoons *y*) + (*y* \rightleftharpoons *x*) = 0

apply (*atomize(full)*)

apply (*intro conjI*; *rule Rep-perm-ext*)

by (*auto simp: Rep-perm-simps fun-eq-iff*)

lemma *swap-self* [*simp*]:

(*x* \rightleftharpoons *x*) = 0

by (*rule Rep-perm-ext, simp add: Rep-perm-simps fun-eq-iff*)

lemma *minus-swap* [*simp*]:

- (*a* \rightleftharpoons *b*) = (*a* \rightleftharpoons *b*)

by (*rule minus-unique [OF swap-cancel(1)]*)

lemma *swap-commute*:

(*a* \rightleftharpoons *b*) = (*b* \rightleftharpoons *a*)

by (*rule Rep-perm-ext*)

(*simp add: Rep-perm-swap fun-eq-iff*)

lemma *swap-triple*:

assumes *a* \neq *b* **and** *c* \neq *b*

shows (*a* \rightleftharpoons *c*) + (*b* \rightleftharpoons *c*) + (*a* \rightleftharpoons *c*) = (*a* \rightleftharpoons *b*)

apply (*rule Rep-perm-ext*)

using *assms* **by** (*auto simp add: Rep-perm-simps fun-eq-iff*)

10 Permutation Types

```

ML <
  structure Equivariance = Named-Thms (
    val name = @{binding eqvt}
    val description = equivariance rules
  )
>

setup Equivariance.setup

Infix syntax for PERMUTE has higher precedence than addition, but lower
than unary minus.

consts PERMUTE :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b (infixr · 75)

locale permutation-type =
  fixes permute :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b
  assumes permute-zero [simp]: permute 0 x = x
  and permute-plus [simp]: permute (p + q) x = permute p (permute q x)
begin

adhoc-overloading
  PERMUTE permute

Equivariance.

definition eqvt :: 'b  $\Rightarrow$  bool
where
  eqvt x  $\longleftrightarrow$  ( $\forall p. p \cdot x = x$ )

definition unpermute :: 'a perm  $\Rightarrow$  'b  $\Rightarrow$  'b
where
  unpermute p = permute ( $-$  p)

definition variants :: ('b  $\times$  'b) set
where
  variants = {(x, y).  $\exists \pi. \pi \cdot x = y$ }

lemma permute-diff [simp]:
  shows (p - q) · x = p · - q · x
  unfolding diff-conv-add-uminus permute-plus by simp

lemma permute-minus-cancel [simp]:
  shows p · - p · x = x
  and - p · p · x = x
  unfolding permute-plus [symmetric] by simp-all

lemma permute-swap-cancel [simp]:
  shows (a  $\rightleftharpoons$  b) · (a  $\rightleftharpoons$  b) · x = x
  unfolding permute-plus [symmetric]

```

```

    by (simp add: swap-cancel)

lemma permute-swap-cancel2 [simp]:
  shows  $(a \rightleftharpoons b) \cdot (b \rightleftharpoons a) \cdot x = x$ 
  unfolding permute-plus [symmetric]
  by (simp add: swap-commute)

lemma inj-permute [simp]:
  shows inj  $((\cdot) \ p)$ 
  by (rule inj-on-inverseI) (rule permute-minus-cancel)

lemma surj-permute [simp]:
  shows surj  $((\cdot) \ p)$ 
  by (rule surjI) (rule permute-minus-cancel)

lemma bij-permute [simp]:
  shows bij  $((\cdot) \ p)$ 
  by (rule bijI [OF inj-permute surj-permute])

lemma inv-permute:
  shows  $inv ((\cdot) \ p) = (\cdot) \ (- \ p)$ 
  by (rule inv-equality) (simp-all)

lemma permute-minus:
  shows  $(\cdot) \ (- \ p) = inv ((\cdot) \ p)$ 
  by (simp add: inv-permute)

lemma permute-eq-iff [simp]:
  shows  $p \cdot x = p \cdot y \longleftrightarrow x = y$ 
  by (rule inj-permute [THEN inj-eq])

lemma variants-refl:
   $(x, x) \in variants$ 
proof -
  have  $0 \cdot x = x$  by simp
  then have  $\exists \pi. \pi \cdot x = x$  ..
  then show ?thesis by (auto simp: variants-def)
qed

lemma variants-sym:
  assumes  $(x, y) \in variants$ 
  shows  $(y, x) \in variants$ 
proof -
  from assms obtain  $\pi$  where  $y = \pi \cdot x$  by (auto simp: variants-def)
  then have  $-\pi \cdot y = x$  by simp
  then show ?thesis by (auto simp: variants-def)
qed

lemma variants-trans:

```

assumes $(x, y) \in \text{variants}$ **and** $(y, z) \in \text{variants}$
shows $(x, z) \in \text{variants}$
proof –
from *assms* **obtain** π_1 **and** π_2
where $y = \pi_1 \cdot x$ **and** $z = \pi_2 \cdot y$ **by** (*auto simp: variants-def*)
then have $(\pi_2 + \pi_1) \cdot x = z$ **by** *simp*
then have $\exists \pi. \pi \cdot x = z$ **..**
then show *?thesis* **by** (*auto simp: variants-def*)
qed

lemma *variants-equiv-on-TRS*:
equiv R (variants \cap $R \times R$)
by (*rule equivI*)
(auto simp: refl-on-def sym-def trans-def variants-refl dest: variants-sym variants-trans)

lemma *variants-TRS*:
equiv UNIV variants
by (*rule equivI*)
(auto simp: refl-on-def sym-def trans-def variants-refl dest: variants-sym variants-trans)

lemma *permute-flip*: $x = \pi \cdot y \implies y = -\pi \cdot x$ **by** *auto*

end

definition *permute-atom* :: $'a \text{ perm} \Rightarrow 'a \Rightarrow 'a$
where
permute-atom p a = (Rep-perm p) a

adhoc-overloading
PERMUTE permute-atom

interpretation *atom-pt*: *permutation-type permute-atom*
by *standard (simp add: permute-atom-def Rep-perm-simps)+*

lemma *swap-atom*:
 $(a \rightleftharpoons b) \cdot c = (\text{if } c = a \text{ then } b \text{ else if } c = b \text{ then } a \text{ else } c)$
by (*simp add: permute-atom-def Rep-perm-swap*)

lemma *swap-atom-simps* [*simp*]:
 $(a \rightleftharpoons b) \cdot a = b$
 $(a \rightleftharpoons b) \cdot b = a$
 $c \neq a \implies c \neq b \implies (a \rightleftharpoons b) \cdot c = c$
by (*simp-all add: swap-atom*)

lemma *perm-eq-iff*:
shows $p = q \iff (\forall a. p \cdot a = q \cdot a)$
unfolding *permute-atom-def*

by (*metis Rep-perm-ext ext*)

definition *permute-perm* :: 'a perm \Rightarrow 'a perm \Rightarrow 'a perm
where
permute-perm *p q* = *p* + *q* + - *p*

adhoc-overloading
PERMUTE permute-perm

interpretation *perm-pt*: *permutation-type permute-perm*
by *standard*
(*simp-all add: permute-perm-def minus-add, simp only: diff-conv-add-uminus add.assoc*)

lemma *permute-self*:
shows *p* • *p* = *p*
by (*simp add: permute-perm-def add.assoc*)

lemma *permute-minus-self*:
shows (-*p*) • *p* = *p*
by (*simp add: add.assoc permute-perm-def*)

lemma (**in** *permutation-type*) *permute-eqt*:
fixes *x* :: 'b
shows *p* • (*q* • *x*) = (*p* • *q*) • (*p* • *x*)
by (*simp add: permute-perm-def*)

lemma *zero-perm-eqt* [*eqt*]:
shows *p* • (0 :: ('a :: infinite) perm) = 0
by (*simp add: permute-perm-def*)

lemma *add-perm-eqt* [*eqt*]:
fixes *p p1 p2* :: ('a :: infinite) perm
shows *p* • (*p1* + *p2*) = *p* • *p1* + *p* • *p2*
by (*simp add: permute-perm-def perm-eq-iff*)

locale *fun-pt* =
dom: *permutation-type perm1* + *ran*: *permutation-type perm2*
for *perm1* :: ('a :: infinite) perm \Rightarrow 'b \Rightarrow 'b
and *perm2* :: 'a perm \Rightarrow 'c \Rightarrow 'c
begin

adhoc-overloading
PERMUTE perm1 perm2

definition *permute-fun* :: 'a perm \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c)
where
permute-fun *p f* = ($\lambda x. p \cdot (f (-p \cdot x))$)

adhoc-overloading

PERMUTE permute-fun

end

sublocale *fun-pt* \subseteq *permutation-type permute-fun*

by *standard (auto simp: permute-fun-def, metis dom.permute-plus minus-add)*

locale *fun-comp-pt* =

a: *fun-pt* *pa pb* +

b: *fun-pt* *pb pc* +

c: *fun-pt* *pa pc*

for *pa* :: ('a :: infinite) *perm* \Rightarrow 'b \Rightarrow 'b

and *pb* :: 'a *perm* \Rightarrow 'c \Rightarrow 'c

and *pc* :: 'a *perm* \Rightarrow 'd \Rightarrow 'd

begin

adhoc-overloading

PERMUTE pa pb pc a.permute-fun b.permute-fun c.permute-fun

lemma *comp-eqvt'*:

fixes *g* :: 'b \Rightarrow 'c **and** *f* :: 'c \Rightarrow 'd

shows $p \cdot (\lambda x. f (g x)) = (\lambda x. (p \cdot f) ((p \cdot g) x))$

by (*simp add: a.permute-fun-def b.permute-fun-def c.permute-fun-def*)

lemma *comp-eqvt [eqvt]*:

fixes *g* :: 'b \Rightarrow 'c **and** *f* :: 'c \Rightarrow 'd

shows $p \cdot (f \circ g) = (p \cdot f) \circ (p \cdot g)$

by (*simp add: comp-def comp-eqvt'*)

end

context *fun-pt*

begin

lemma *apply-eqvt*:

fixes *f* :: 'b \Rightarrow 'c **and** *x* :: 'b

shows $p \cdot (f x) = (p \cdot f) (p \cdot x)$

by (*simp add: permute-fun-def*)

lemma *lambda-eqvt*:

fixes *p* :: ('a::infinite) *perm* **and** *f* :: 'b \Rightarrow 'c

shows $p \cdot f = (\lambda x. p \cdot (f (dom.unpermute p x)))$

by (*simp add: permute-fun-def dom.unpermute-def*)

lemma *unpermute-self*:

$p \cdot unpermute p x = x$

by (*simp add: unpermute-def*)

```

lemma permute-fun-comp:
  fixes  $p :: 'a \text{ perm}$ 
  shows  $p \cdot f = ((\cdot) p) \circ f \circ ((\cdot) (-p))$ 
  by (simp add: comp-def permute-fun-def)

end

definition permute-bool ::  $'a \text{ perm} \Rightarrow \text{bool} \Rightarrow \text{bool}$  where
  permute-bool  $p \ b = b$ 

adhoc-overloading
  PERMUTE permute-bool

interpretation bool-pt: permutation-type permute-bool
  by standard (simp add: permute-bool-def)+

lemma permute-boolE:
  fixes  $p :: ('a :: \text{infinite}) \text{ perm}$ 
  shows  $p \cdot P \Longrightarrow P$ 
  by (simp add: permute-bool-def)

lemma permute-boolI:
  fixes  $p :: ('a :: \text{infinite}) \text{ perm}$ 
  shows  $P \Longrightarrow p \cdot P$ 
  by (simp add: permute-bool-def)

lemma Not-eqvt [eqvt]:
   $p \cdot (\neg A) \longleftrightarrow \neg (p \cdot A)$ 
  by (simp add: permute-bool-def)

lemma conj-eqvt [eqvt]:
   $p \cdot (A \wedge B) \longleftrightarrow (p \cdot A) \wedge (p \cdot B)$ 
  by (simp add: permute-bool-def)

lemma imp-eqvt [eqvt]:
   $p \cdot (A \longrightarrow B) \longleftrightarrow (p \cdot A) \longrightarrow (p \cdot B)$ 
  by (simp add: permute-bool-def)

lemmas
  True-eqvt [eqvt] = permute-bool-def [of - True] and
  False-eqvt [eqvt] = permute-bool-def [of - False]

lemma disj-eqvt [eqvt]:
   $p \cdot (A \vee B) \longleftrightarrow (p \cdot A) \vee (p \cdot B)$ 
  by (simp add: permute-bool-def)

locale pred-pt =
  arg: permutation-type perm

```

```

for  $perm :: ('a :: infinite) \Rightarrow 'b \Rightarrow 'b$ 
begin

definition
   $permute\_pred :: ('a :: infinite) \Rightarrow 'b \Rightarrow bool \Rightarrow ('b \Rightarrow bool)$ 
where
   $permute\_pred\ p\ P = (\lambda x. P\ (perm\ (-p)\ x))$ 

end

sublocale  $pred\_pt \subseteq fun\_pt\ perm\ permute\_bool$ 
  rewrites  $permute\_fun = permute\_pred$ 
proof –
  show *:  $fun\_pt\ perm\ permute\_bool ..$ 
  show  $fun\_pt.permute\_fun\ perm\ permute\_bool = permute\_pred$ 
  unfolding  $fun\_pt.permute\_fun\_def\ [OF\ *,\ abs\_def]\ permute\_bool\_def\ permute\_pred\_def\ [abs\_def]$ 
  ..
qed

definition  $permute\_atom\_pred :: ('a :: infinite) \Rightarrow 'a \Rightarrow bool \Rightarrow 'a \Rightarrow bool$ 
where
   $permute\_atom\_pred\ p\ P = (\lambda x. P\ (-p \cdot x))$ 

adhoc-overloading
   $PERMUTE\ permute\_atom\_pred$ 

interpretation  $atom\_pred\_pt: pred\_pt\ permute\_atom$ 
  rewrites  $pred\_pt.permute\_pred\ permute\_atom = permute\_atom\_pred$ 
proof –
  show *:  $pred\_pt\ permute\_atom ..$ 
  show  $pred\_pt.permute\_pred\ permute\_atom = permute\_atom\_pred$ 
  by ( $simp\ add: permute\_atom\_pred\_def\ [abs\_def]\ pred\_pt.permute\_pred\_def\ [OF\ *,\ abs\_def]$ )
qed

context  $permutation\_type$ 
begin

interpretation  $pred: pred\_pt\ permute ..$ 

adhoc-overloading
   $PERMUTE\ pred.permute\_pred$ 

lemma  $eq\_eqvt\ [eqvt]:$ 
  fixes  $x :: 'b$ 
  shows  $p \cdot (x = y) \longleftrightarrow (p \cdot x) = (p \cdot y)$ 
  unfolding  $permute\_eq\_iff\ permute\_bool\_def ..$ 

```

lemma *all-eqv* [*eqv*]:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\forall x. P x) = (\forall x. (p \cdot P) x)$
unfolding *pred.permute-pred-def permute-bool-def*
by (*metis permute-plus permute-zero right-minus*)

lemma *all-eqv'*:
fixes $P :: 'c \Rightarrow bool$
shows $p \cdot (\forall x. P x) = (\forall x. p \cdot (P x))$
by (*simp add: permute-bool-def*)

lemma *ball-eqv'*:
fixes $P :: 'c \Rightarrow bool$
shows $p \cdot (\forall x \in A. P x) = (\forall x \in A. p \cdot (P x))$
by (*simp add: permute-bool-def*)

lemma *ex-eqv* [*eqv*]:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\exists x. P x) = (\exists x. (p \cdot P) x)$
unfolding *Ex-def pred.permute-pred-def permute-bool-def*
by (*simp*) (*metis permute-minus-cancel(2)*)

lemma *ex1-eqv* [*eqv*]:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\exists !x. P x) = (\exists !x. (p \cdot P) x)$
unfolding *Ex1-def pred-pt.permute-pred-def*
by (*simp add: eqv pred.permute-pred-def*)
(metis permute-minus-cancel(2))

lemma *if-eqv* [*eqv*]:
fixes $x :: 'b$
shows $p \cdot (\text{if } b \text{ then } x \text{ else } y) = (\text{if } p \cdot b \text{ then } p \cdot x \text{ else } p \cdot y)$
by (*simp add: pred-pt.permute-pred-def permute-bool-def*)

lemma *all-eqv2*:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\forall x. P x) = (\forall x. p \cdot P (- p \cdot x))$
by (*simp add: eqv pred.permute-pred-def*)
(metis permute-bool-def)

lemma *ex-eqv2*:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\exists x. P x) = (\exists x. p \cdot P (- p \cdot x))$
by (*simp add: eqv pred.permute-pred-def*)
(metis permute-bool-def)

lemma *ex1-eqv2*:
fixes $P :: 'b \Rightarrow bool$
shows $p \cdot (\exists !x. P x) = (\exists !x. p \cdot P (- p \cdot x))$

```

    by (simp add: eqvt pred.permute-pred-def)
      (metis permute-bool-def)

end

locale set-pt =
  elt: permutation-type perm for perm :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b
begin

ad hoc overloading
  PERMUTE perm

definition permute-set :: 'a perm  $\Rightarrow$  'b set  $\Rightarrow$  'b set
where
  permute-set p A = {p · x | x. x ∈ A}

ad hoc overloading
  PERMUTE permute-set

lemma permute-set-subset:
  fixes  $\pi :: ('a :: infinite) perm$ 
  and A :: 'b set
  assumes A  $\subseteq$  B
  shows  $\pi \cdot A \subseteq \pi \cdot B$ 
  using assms by (auto simp: permute-set-def)

lemma subset-imp-ex-perm:
  fixes A :: 'b set
  assumes A  $\subseteq$  B
  shows  $\forall x \in A. \exists p. \exists y \in B. p \cdot x = y$ 
  using assms by (auto) (metis elt.permute-zero set-rev-mp)

end

sublocale set-pt  $\subseteq$  permutation-type permute-set
  by standard (auto simp: permute-set-def)

context set-pt
begin

lemma permute-set-eq:
  p · X = {x. - p · x ∈ X}
  by (auto simp: permute-set-def) (metis elt.permute-minus-cancel(1))

lemma permute-set-eq-image:
  p · X = (·) p ‘ X
  by (auto simp: permute-set-def)

lemma permute-set-eq-vimage:

```

$p \cdot X = (\cdot) (- p) - ' X$
by (*simp add: permute-set-eq vimage-def*)

lemma *permute-finite* [*simp*]:
 $finite (p \cdot X) = finite X$
unfolding *permute-set-eq-vimage*
using *bij-permute* **by** (*metis elt.bij-permute finite-vimage-iff*)

lemma *mem-permute-iff*:
fixes $p :: 'a \text{ perm}$
shows $(p \cdot x) \in (p \cdot X) \longleftrightarrow x \in X$
by (*auto simp: permute-set-def*)

lemma *inv-mem-simps* [*simp*]:
fixes $p :: 'a \text{ perm}$
shows $(-p \cdot x) \in X \longleftrightarrow x \in (p \cdot X)$
and $x \in (-p \cdot X) \longleftrightarrow (p \cdot x) \in X$
by (*metis permute-minus-cancel(2) mem-permute-iff*)+

lemma *empty-eqv* [*simp*]:
 $p \cdot \{\} = \{\}$
by (*simp add: permute-set-def*)

lemma *permute-set-emptyD* [*dest*]:
 $p \cdot A = \{\} \implies A = \{\}$
by (*simp add: permute-set-def*)

lemma *insert-eqv* [*eqvt*]:
 $p \cdot (insert\ x\ A) = insert\ (p \cdot x)\ (p \cdot A)$
unfolding *permute-set-eq-image image-insert ..*

lemma *mem-eqv* [*eqvt*]:
shows $p \cdot (x \in A) \longleftrightarrow (p \cdot x) \in (p \cdot A)$
unfolding *permute-bool-def permute-set-def*
by (*simp add: eqvt*)

interpretation *elt-pred-pt*: *pred-pt perm ..*

adhoc-overloading
 $PERMUTE\ elt\text{-}pred\text{-}pt.\text{permute-pred}$

lemma *Collect-eqv* [*eqvt*]:
 $p \cdot \{x. P\ x\} = \{x. (p \cdot P)\ x\}$
by (*simp add: permute-set-eq elt-pred-pt.permute-pred-def*)

lemma *inter-eqv* [*eqvt*]:
 $p \cdot (A \cap B) = (p \cdot A) \cap (p \cdot B)$
unfolding *Int-def permute-set-eq* **by** *simp*

lemma *Bex-eqv* [eqvt]:

$$p \cdot (\exists x \in S. P x) = (\exists x \in (p \cdot S). (p \cdot P) x)$$

by (*simp add: Bex-def pred-pt.permute-pred-def eqvt elt-pred-pt.permute-pred-def permute-set-def*)

lemma *Ball-eqv* [eqvt]:

$$p \cdot (\forall x \in S. P x) = (\forall x \in (p \cdot S). (p \cdot P) x)$$

by (*simp add: Ball-def eqvt permute-set-def elt-pred-pt.permute-pred-def*)

lemma *UNIV-eqv* [eqvt]:

$$p \cdot \text{UNIV} = \text{UNIV}$$

unfolding *UNIV-def* **by** (*auto simp add: permute-set-def*) (*metis elt.permute-minus-cancel(1)*)

lemma *union-eqv* [eqvt]:

$$p \cdot (A \cup B) = (p \cdot A) \cup (p \cdot B)$$

by (*auto simp: Un-def permute-set-def*)

lemma *Union-eqv* [eqvt]:

$$p \cdot \bigcup A = \bigcup ((\cdot) p \cdot A)$$

by (*auto simp: permute-set-def*)

lemma *UNION-eqv* [eqvt]:

$$p \cdot (\bigcup (f \cdot A)) = \bigcup ((\lambda x. p \cdot f x) \cdot A)$$

by (*auto simp: permute-set-def*)

lemma *Diff-eqv* [eqvt]:

fixes *A B* :: 'b set

$$\text{shows } p \cdot (A - B) = (p \cdot A) - (p \cdot B)$$

by (*auto simp: set-diff-eq permute-set-def*)

lemma *Compl-eqv* [eqvt]:

fixes *A* :: 'b set

$$\text{shows } p \cdot (- A) = - (p \cdot A)$$

by (*auto simp: permute-set-def Compl-eq-Diff-UNIV*)
(*metis elt.permute-minus-cancel(1)*)

lemma *subset-eqv* [eqvt]:

$$p \cdot (S \subseteq T) \longleftrightarrow (p \cdot S) \subseteq (p \cdot T)$$

by (*simp add: subset-eq eqvt elt-pred-pt.permute-pred-def*)

lemma *psubset-eqv* [eqvt]:

$$p \cdot (S \subset T) \longleftrightarrow (p \cdot S) \subset (p \cdot T)$$

by (*simp add: psubset-eq eqvt*)

end

definition *permute-atom-set* :: ('a :: infinite) perm \Rightarrow 'a set \Rightarrow 'a set

where

$$\text{permute-atom-set } p \ A = \{p \cdot x \mid x. x \in A\}$$

adhoc-overloading

PERMUTE permute-atom-set

interpretation *atom-set-pt: set-pt permute-atom*

rewrites *set-pt.permute-set permute-atom = permute-atom-set*

and *pred-pt.permute-pred permute-atom = permute-atom-pred*

proof –

show **: set-pt permute-atom ..*

show *set-pt.permute-set permute-atom = permute-atom-set*

by (*simp add: permute-atom-set-def [abs-def] set-pt.permute-set-def [OF *, abs-def]*)

have *** : pred-pt permute-atom ..*

show *pred-pt.permute-pred permute-atom = permute-atom-pred*

by (*simp add: permute-atom-pred-def [abs-def] pred-pt.permute-pred-def [OF **, abs-def]*)

qed

lemma *swap-set-not-in:*

assumes *a ∉ S b ∉ S*

shows *(a ⇌ b) · S = S*

using *assms by (auto simp: permute-atom-set-def swap-atom)*

lemma *swap-set-in:*

assumes *a ∈ S b ∉ S*

shows *(a ⇌ b) · S ≠ S*

using *assms by (force simp: permute-atom-set-def swap-atom)*

lemma *swap-set-in-eq:*

assumes *a ∈ S b ∉ S*

shows *(a ⇌ b) · S = (S − {a}) ∪ {b}*

using *assms by (auto simp: permute-atom-set-def swap-atom)*

lemma *swap-set-both-in:*

assumes *a ∈ S b ∈ S*

shows *(a ⇌ b) · S = S*

using *assms by (auto simp add: permute-atom-set-def swap-atom) metis*

definition *permute-unit :: 'a perm ⇒ unit ⇒ unit*

where

permute-unit p (u :: unit) = u

interpretation *unit-pt: permutation-type permute-unit*

by *standard (simp add: permute-unit-def)+*

adhoc-overloading

PERMUTE permute-unit

locale *prod-pt =*

```

    fst: permutation-type perm1 + snd: permutation-type perm2
  for perm1 :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b
  and perm2 :: 'a perm  $\Rightarrow$  'c  $\Rightarrow$  'c
begin

adhoc-overloading
  PERMUTE perm1 perm2

fun permute-prod :: 'a perm  $\Rightarrow$  ('b  $\times$  'c)  $\Rightarrow$  ('b  $\times$  'c)
where
  permute-prod-eqvt [eqvt]: permute-prod p (x, y) = (p  $\cdot$  x, p  $\cdot$  y)

adhoc-overloading
  PERMUTE permute-prod

declare permute-prod.simps [simp del]

end

sublocale prod-pt  $\subseteq$  permutation-type permute-prod
  by standard (auto simp: eqvt)

locale sum-pt =
  l: permutation-type perm1 + r: permutation-type perm2
  for perm1 :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b
  and perm2 :: 'a perm  $\Rightarrow$  'c  $\Rightarrow$  'c
begin

adhoc-overloading
  PERMUTE perm1 perm2

fun permute-sum :: 'a perm  $\Rightarrow$  ('b + 'c)  $\Rightarrow$  ('b + 'c)
where
  permute-sum p (Inl x) = Inl (p  $\cdot$  x) |
  permute-sum p (Inr y) = Inr (p  $\cdot$  y)

adhoc-overloading
  PERMUTE permute-sum

end

sublocale sum-pt  $\subseteq$  permutation-type permute-sum
  apply unfold-locales
  subgoal for x by (cases x, auto)
  subgoal for p q x by (cases x, auto)
  done

locale list-pt =
  elt: permutation-type perm for perm :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b

```

```

begin

adhoc-overloading
  PERMUTE perm

definition permute-list :: 'a perm  $\Rightarrow$  'b list  $\Rightarrow$  'b list
where
  [simp]: permute-list p = map ( $\lambda x. p \cdot x$ )

adhoc-overloading
  PERMUTE permute-list

lemma nth-eqv:
   $i < \text{length } xs \implies \pi \cdot (xs ! i) = (\pi \cdot xs) ! i$ 
  by simp

end

sublocale list-pt  $\subseteq$  permutation-type permute-list
  apply unfold-locales
  subgoal for x by (cases x, auto)
  subgoal for p q x by (cases x, auto)
  done

locale option-pt =
  elt: permutation-type perm for perm :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b
begin

adhoc-overloading
  PERMUTE perm

fun permute-option :: 'a perm  $\Rightarrow$  'b option  $\Rightarrow$  'b option
where
  permute-option p None = None |
  permute-option p (Some x) = Some (p  $\cdot$  x)

adhoc-overloading
  PERMUTE permute-option

end

sublocale option-pt  $\subseteq$  permutation-type permute-option
  apply unfold-locales
  subgoal for x by (cases x, auto)
  subgoal for p q x by (cases x, auto)
  done

locale rel-pt =
  step: prod-pt perm perm for perm :: ('a :: infinite perm)  $\Rightarrow$  'b  $\Rightarrow$  'b

```

```

begin

adhoc-overloading
  PERMUTE perm step.permute-prod

interpretation set-pt step.permute-prod ..

adhoc-overloading
  PERMUTE permute-set

end

sublocale rel-pt  $\subseteq$  set-pt step.permute-prod ..

context rel-pt
begin

lemma relcomp-eqvt:
  fixes  $R\ S :: 'b\ rel$ 
  assumes  $\bigwedge p. p \cdot R = R$  and  $\bigwedge p. p \cdot S = S$ 
  shows  $p \cdot (R\ O\ S) = R\ O\ S$ 
proof -
  interpret step-pred: pred-pt step.permute-prod ..
  { fix  $a\ b\ x\ y\ z$  assume  $-p \cdot (a, b) = (x, z)$  and  $(a, y) \in R$  and  $(y, b) \in S$ 
    moreover then have  $-p \cdot (a, y) \in R$  and  $-p \cdot (y, b) \in S$ 
    by (simp-all, auto simp: eqvt assms)
    ultimately have  $(x, -p \cdot y) \in R$  and  $(-p \cdot y, z) \in S$  by (auto simp: eqvt)
    then have  $\exists y. (x, y) \in R \wedge (y, z) \in S$  by blast }
  moreover
  { fix  $a\ b\ x\ y\ z$  assume  $-p \cdot (a, b) = (x, z)$  and  $(x, y) \in R$  and  $(y, z) \in S$ 
    moreover then have  $p \cdot (x, y) \in R$  and  $p \cdot (y, z) \in S$ 
    by (auto simp: inv-mem-simps [symmetric] assms)
    ultimately have  $(a, p \cdot y) \in R$  and  $(p \cdot y, b) \in S$  by (auto simp: eqvt)
    then have  $\exists y. (a, y) \in R \wedge (y, b) \in S$  by blast }
  moreover have  $R\ O\ S = \{(x, z). \exists y. (x, y) \in R \wedge (y, z) \in S\}$  by auto
  ultimately have  $p \cdot (R\ O\ S) = \{(x, z). \exists y. p \cdot ((x, y) \in R) \wedge p \cdot ((y, z) \in S)\}$ 
    using [[show-variants]]
    by (auto simp: eqvt step-pred.permute-pred-def mem-permute-iff simp del: step.permute-prod-eqvt)
  then show ?thesis by (auto simp: permute-bool-def)
qed

end

```

11 Pure Types

```

locale pure =
  permutation-type permute for permute :: ('a :: infinite) perm  $\Rightarrow$  'b  $\Rightarrow$  'b +
  assumes permute-pure [simp]:  $(p :: 'a\ perm) \cdot (x :: 'b) = x$ 

```

interpretation *unit-pure: pure permute-unit*
by *standard (simp add: permute-unit-def)*

interpretation *bool-pure: pure permute-bool*
by *standard (simp add: permute-bool-def)*

lemma (*in fun-pt*) *eqvt-fun-iff*:
 $eqvt\ f \longleftrightarrow (\forall (p :: 'a\ perm)\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
by (*auto simp add: eqvt-def permute-fun-def*)
(metis dom.unpermute-def apply-eqvt lambda-eqvt)

lemma *bool-pt-eqvt [simp]*:
 $bool-pt.eqvt\ TYPE('a :: infinite)\ x$
by (*simp add: bool-pt.eqvt-def permute-bool-def*)

lemma *swap-eqvt [eqvt]*:
fixes $p :: ('a :: infinite)\ perm$
shows $p \cdot (a \rightleftharpoons b) = (p \cdot a \rightleftharpoons p \cdot b)$
by (*auto simp add: permute-perm-def swap-atom perm-eq-iff*)

consts
 $FRESH :: 'a \Rightarrow 'b \Rightarrow bool$ (**infix** $\#$ 55)

context *permutation-type*
begin

The support of x (aka, the set of free variables, provided we have infinitely many atoms at our disposal).

definition $supp :: 'b \Rightarrow 'a\ set$
where
 $supp\ x = \{a.\ infinite\ \{b.\ (a \rightleftharpoons b) \cdot x \neq x\}\}$

definition $fresh :: 'a \Rightarrow 'b \Rightarrow bool$
where
 $fresh\ a\ x \longleftrightarrow a \notin supp\ x$

adhoc-overloading
 $FRESH\ fresh$

definition $fresh-set :: 'a\ set \Rightarrow 'b \Rightarrow bool$
where
 $fresh-set\ A\ x \longleftrightarrow (\forall a \in A.\ a \# x)$

adhoc-overloading
 $FRESH\ fresh-set$

definition $supports :: 'a\ set \Rightarrow 'b \Rightarrow bool$
where
 $supports\ S\ x \longleftrightarrow (\forall a\ b.\ (a \notin S \wedge b \notin S \longrightarrow (a \rightleftharpoons b) \cdot x = x))$

```

lemma fresh-set-disjoint:
  assumes  $A \nparallel x$ 
  shows  $A \cap \text{supp } x = \{\}$ 
  using assms unfolding fresh-set-def fresh-def
  by (metis disjoint-iff-not-equal)

lemma supp-is-subset:
  fixes  $S :: 'a \text{ set}$ 
  and  $x :: 'b$ 
  assumes  $a1: \text{supports } S \ x$ 
  and  $a2: \text{finite } S$ 
  shows  $\text{supp } x \subseteq S$ 
proof (rule ccontr)
  assume  $\neg (\text{supp } x \subseteq S)$ 
  then obtain  $a$  where  $b1: a \in \text{supp } x$  and  $b2: a \notin S$  by auto
  from  $a1 \ b2$  have  $\forall b. b \notin S \longrightarrow (a \rightleftharpoons b) \cdot x = x$  unfolding supports-def by
auto
  then have  $\{b. (a \rightleftharpoons b) \cdot x \neq x\} \subseteq S$  by auto
  with  $a2$  have finite  $\{b. (a \rightleftharpoons b) \cdot x \neq x\}$  by (simp add: finite-subset)
  then have  $a \notin (\text{supp } x)$  unfolding supp-def by simp
  with  $b1$  show False by simp
qed

lemma supp-conv-fresh:
   $\text{supp } x = \{a. \neg a \nparallel x\}$ 
  by (simp add: fresh-def)

lemma swap-rel-trans:
  fixes  $a \ b \ c :: 'a$  and  $x :: 'b$ 
  assumes  $(a \rightleftharpoons c) \cdot x = x$  and  $(b \rightleftharpoons c) \cdot x = x$ 
  shows  $(a \rightleftharpoons b) \cdot x = x$ 
proof (cases)
  assume  $a = b \vee c = b$ 
  with assms show  $(a \rightleftharpoons b) \cdot x = x$  by auto
next
  assume  $*$ :  $\neg (a = b \vee c = b)$ 
  have  $((a \rightleftharpoons c) + (b \rightleftharpoons c) + (a \rightleftharpoons c)) \cdot x = x$ 
  using assms by simp
  also have  $(a \rightleftharpoons c) + (b \rightleftharpoons c) + (a \rightleftharpoons c) = (a \rightleftharpoons b)$ 
  using assms  $*$  by (simp add: swap-triple)
  finally show  $(a \rightleftharpoons b) \cdot x = x$  .
qed

lemma obtain-atom:
  fixes  $X :: 'a \text{ set}$ 
  assumes  $X: \text{finite } X$ 
  obtains  $a$  where  $a \notin X$ 
proof –

```

```

from  $X$  have  $MOST\ a.\ a \notin X$ 
  unfolding  $MOST\text{-}iff\text{-}cofinite$  by  $simp$ 
then have  $INFM\ a.\ a \notin X$  using  $infinite\text{-}UNIV$  and  $X$  by ( $metis\ Collect\text{-}mem\text{-}eq$ 
 $INFM\text{-}iff\text{-}infinite\ finite\text{-}Collect\text{-}not$ )
  then obtain  $a$  where  $a \notin X$ 
  by ( $auto\ elim:\ INFM\text{-}E$ )
  then show  $?thesis\ ..$ 
qed

```

```

lemma  $swap\text{-}fresh\text{-}fresh$ :
  assumes  $a:\ a \# x$  and  $b:\ b \# x$ 
  shows  $(a \rightleftharpoons b) \cdot x = x$ 
proof  $-$ 
  have  $finite\ \{c.\ (a \rightleftharpoons c) \cdot x \neq x\}$   $finite\ \{c.\ (b \rightleftharpoons c) \cdot x \neq x\}$ 
    using  $a\ b$  unfolding  $fresh\text{-}def\ supp\text{-}def$  by  $simp\text{-}all$ 
  then have  $finite\ (\{c.\ (a \rightleftharpoons c) \cdot x \neq x\} \cup \{c.\ (b \rightleftharpoons c) \cdot x \neq x\})$  by  $simp$ 
  then obtain  $c$ 
    where  $(a \rightleftharpoons c) \cdot x = x\ (b \rightleftharpoons c) \cdot x = x$ 
    by ( $rule\ obtain\text{-}atom$ ) ( $auto$ )
  then show  $(a \rightleftharpoons b) \cdot x = x$  by ( $rule\ swap\text{-}rel\text{-}trans$ )
qed

```

The notion of support does not make sense for a finite set of atoms.

```

lemma  $supp\text{-}empty$ :
  assumes  $finite\ (UNIV :: 'a\ set)$ 
  shows  $supp\ x = \{\}$ 
  using  $assms$  by ( $auto\ simp:\ supp\text{-}def$ ) ( $metis\ rev\text{-}finite\text{-}subset\ top\text{-}greatest$ )

```

```

lemma  $fresh\text{-}ex$ :
  assumes  $finite\ (supp\ x)$ 
  shows  $\exists a::'a.\ a \# x$ 
  using  $ex\text{-}new\text{-}if\text{-}finite$  [ $OF\ infinite\text{-}UNIV\ assms$ ] by ( $simp\ add:\ fresh\text{-}def$ )

```

```

lemma  $fresh\text{-}set\text{-}supp\text{-}conv$ :
  shows  $supp\ x \# y \implies supp\ y \# x$ 
  by ( $auto\ simp\ add:\ fresh\text{-}set\text{-}def\ fresh\text{-}def$ )

```

```

lemma  $supp\text{-}supports$ :
  fixes  $x :: 'b$ 
  shows  $supports\ (supp\ x)\ x$ 
unfolding  $supports\text{-}def$ 
proof ( $intro\ strip$ )
  fix  $a\ b$ 
  assume  $a \notin (supp\ x) \wedge b \notin (supp\ x)$ 
  then have  $a \# x$  and  $b \# x$  by ( $simp\text{-}all\ add:\ fresh\text{-}def$ )
  then show  $(a \rightleftharpoons b) \cdot x = x$  by ( $simp\ add:\ swap\text{-}fresh\text{-}fresh$ )
qed

```

```

lemma  $supp\text{-}is\text{-}least\text{-}supports$ :

```

```

fixes  $S :: 'a \text{ set}$ 
  and  $x :: 'b$ 
assumes  $a1: \text{supports } S \ x$ 
  and  $a2: \text{finite } S$ 
  and  $a3: \bigwedge S'. \text{finite } S' \implies \text{supports } S' \ x \implies S \subseteq S'$ 
shows  $(\text{supp } x) = S$ 
proof (rule equalityI)
  show  $(\text{supp } x) \subseteq S$  using  $a1 \ a2$  by (rule supp-is-subset)
  with  $a2$  have  $\text{fin}: \text{finite } (\text{supp } x)$  by (rule rev-finite-subset)
  have  $\text{supports } (\text{supp } x) \ x$  by (rule supp-supports)
  with  $\text{fin } a3$  show  $S \subseteq \text{supp } x$  by blast
qed

lemma subsetCI:
   $(\bigwedge x. x \in A \implies x \notin B \implies \text{False}) \implies A \subseteq B$  by auto

lemma finite-supp-unique:
  assumes  $a1: \text{supports } S \ x$ 
  assumes  $a2: \text{finite } S$ 
  assumes  $a3: \bigwedge a \ b. [a \in S; b \notin S] \implies (a \rightleftharpoons b) \cdot x \neq x$ 
  shows  $\text{supp } x = S$ 
  using  $a1 \ a2$ 
proof (rule supp-is-least-supports)
  fix  $S'$ 
  assume  $\text{finite } S' \text{ and } \text{supports } S' \ x$ 
  show  $S \subseteq S'$ 
  proof (rule subsetCI)
    fix  $a$ 
    assume  $a \in S \text{ and } a \notin S'$ 
    have  $\text{finite } (S \cup S')$ 
    using  $\langle \text{finite } S \rangle \langle \text{finite } S' \rangle$  by simp
    then obtain  $b$  where  $b \notin S \cup S'$  by (rule obtain-atom)
    then have  $b \notin S \text{ and } b \notin S'$  by simp-all
    then have  $(a \rightleftharpoons b) \cdot x = x$ 
    using  $\langle a \notin S' \rangle \langle \text{supports } S' \ x \rangle$  by (simp add: supports-def)
    moreover have  $(a \rightleftharpoons b) \cdot x \neq x$ 
    using  $\langle a \in S \rangle \langle b \notin S \rangle$  by (rule a3)
    ultimately show  $\text{False}$  by simp
  qed
qed

end

lemma perm-swap-eq:
   $(a \rightleftharpoons b) \cdot p = p \longleftrightarrow (p \cdot (a \rightleftharpoons b)) = (a \rightleftharpoons b)$ 
  unfolding permute-perm-def by (metis add-diff-cancel minus-perm-def)

lemma supports-perm:
   $\text{perm-pt.supports } \{a. p \cdot a \neq a\} \ p$ 

```

```

by (simp add: perm-pt.supports-def perm-swap-eq eqvt)

lemma finite-perm-lemma:
  fixes p :: ('a::infinite) perm
  shows finite {a :: 'a. p · a ≠ a}
  using finite-Rep-perm [of p]
  unfolding permute-atom-def .

lemma supp-perm:
  perm-pt.supp p = {a. p · a ≠ a}
  apply (intro perm-pt.finite-supp-unique supports-perm finite-perm-lemma)
  apply (simp add: perm-swap-eq)
  apply (auto simp: perm-eq-iff swap-atom eqvt)
done

lemma supp-swap:
  perm-pt.supp (a ⇔ b) = (if a = b then {} else {a, b})
  by (auto simp add: supp-perm swap-atom)

lemma supp-zero-perm:
  perm-pt.supp 0 = {}
  by (simp add: supp-perm)

lemma finite-supp-perm:
  finite (perm-pt.supp p)
  by (metis finite-perm-lemma supp-perm)

lemma plus-perm-eq:
  assumes perm-pt.supp p ∩ perm-pt.supp q = {}
  shows p + q = q + p
  unfolding perm-eq-iff
  proof
    fix a :: 'a
    show (p + q) · a = (q + p) · a
    proof -
      { assume a ∉ perm-pt.supp p a ∉ perm-pt.supp q
        then have (p + q) · a = (q + p) · a
          by (simp add: supp-perm)
      }
      moreover
      { assume a: a ∈ perm-pt.supp p a ∉ perm-pt.supp q
        then have p · a ∈ perm-pt.supp p by (simp add: supp-perm)
        then have p · a ∉ perm-pt.supp q using assms by auto
        with a have (p + q) · a = (q + p) · a
          by (simp add: supp-perm)
      }
      moreover
      { assume a: a ∉ perm-pt.supp p a ∈ perm-pt.supp q
        then have q · a ∈ perm-pt.supp q by (simp add: supp-perm)

```

```

    then have  $q \cdot a \notin \text{perm-pt.supp } p$  using assms by auto
    with a have  $(p + q) \cdot a = (q + p) \cdot a$ 
    by (simp add: supp-perm)
  }
  ultimately show  $(p + q) \cdot a = (q + p) \cdot a$ 
  using assms by blast
qed
qed

```

```

lemma supp-plus-perm:
   $\text{perm-pt.supp } (p + q) \subseteq \text{perm-pt.supp } p \cup \text{perm-pt.supp } q$ 
  by (auto simp add: supp-perm)

```

```

lemma supp-plus-perm-eq:
  assumes  $\text{perm-pt.supp } p \cap \text{perm-pt.supp } q = \{\}$ 
  shows  $\text{perm-pt.supp } (p + q) = \text{perm-pt.supp } p \cup \text{perm-pt.supp } q$ 
proof -
  { fix a
    assume  $a \in \text{perm-pt.supp } p$ 
    then have  $a \notin \text{perm-pt.supp } q$  using assms by auto
    then have  $a \in \text{perm-pt.supp } (p + q)$  using  $\langle a \in \text{perm-pt.supp } p \rangle$ 
    by (simp add: supp-perm)
  }
  moreover
  { fix a
    assume  $a \in \text{perm-pt.supp } q$ 
    then have  $a \notin \text{perm-pt.supp } p$  using assms by auto
    then have  $a \in \text{perm-pt.supp } (q + p)$  using  $\langle a \in \text{perm-pt.supp } q \rangle$ 
    by (simp add: supp-perm)
    then have  $a \in \text{perm-pt.supp } (p + q)$  using assms plus-perm-eq
    by metis
  }
  ultimately have  $\text{perm-pt.supp } p \cup \text{perm-pt.supp } q \subseteq \text{perm-pt.supp } (p + q)$ 
  by blast
  then show  $\text{perm-pt.supp } (p + q) = \text{perm-pt.supp } p \cup \text{perm-pt.supp } q$ 
  using supp-plus-perm
  by blast
qed

```

```

lemma atom-set-avoiding-aux:
  fixes As Xs ::  $('a::\text{infinite}) \text{ set}$ 
  assumes  $b: Xs \subseteq As$ 
  and  $c: \text{finite } As$ 
  shows  $\exists (p::('a::\text{infinite}) \text{ perm}). (p \cdot Xs) \cap As = \{\} \wedge \text{perm-pt.supp } p = (Xs \cup (p \cdot Xs))$ 
proof -
  from b c have finite Xs by (rule finite-subset)
  then show ?thesis using b
  proof (induct rule: finite-subset-induct)

```

```

case empty
have  $0 \cdot \{\} \cap As = \{\}$  by (simp)
moreover
have  $\text{perm-pt.supp } 0 = \{\} \cup 0 \cdot \{\}$  by (simp add: supp-zero-perm)
ultimately show ?case by blast
next
case (insert x Xs)
then obtain p where
  p1:  $(p \cdot Xs) \cap As = \{\}$  and
  p2:  $\text{perm-pt.supp } p = (Xs \cup (p \cdot Xs))$  by blast
from  $\langle x \in As \rangle$  p1 have  $x \notin p \cdot Xs$  by fast
with  $\langle x \notin Xs \rangle$  p2 have  $x \notin \text{perm-pt.supp } p$  by fast
then have px:  $p \cdot x = x$  unfolding supp-perm by simp
have finite  $(As \cup p \cdot Xs \cup \text{perm-pt.supp } p)$ 
  using  $\langle \text{finite } As \rangle \langle \text{finite } Xs \rangle$ 
  by (simp add: set-pt.permute-set-eq-image finite-supp-perm)
then obtain y where  $y \notin (As \cup p \cdot Xs \cup \text{perm-pt.supp } p)$ 
  by (rule atom-set-pt.obtain-atom)
then have y:  $y \notin As$   $y \notin p \cdot Xs$   $y \notin \text{perm-pt.supp } p$ 
  by simp-all
then have py:  $p \cdot y = y$   $x \neq y$  using  $\langle x \in As \rangle$ 
  by (auto simp add: supp-perm)
let ?q =  $(x \rightleftharpoons y) + p$ 
have q:  $?q \cdot \text{insert } x Xs = \text{insert } y (p \cdot Xs)$ 
  unfolding atom-set-pt.insert-eqv
  using  $\langle p \cdot x = x \rangle$ 
  using  $\langle x \notin p \cdot Xs \rangle \langle y \notin p \cdot Xs \rangle$ 
  by (simp add: swap-atom swap-set-not-in)
have  $?q \cdot \text{insert } x Xs \cap As = \{\}$ 
  using  $\langle y \notin As \rangle \langle p \cdot Xs \cap As = \{\} \rangle$ 
  unfolding q by simp
moreover
have  $\text{perm-pt.supp } (x \rightleftharpoons y) \cap \text{perm-pt.supp } p = \{\}$  using px py
  unfolding supp-swap by (simp add: supp-perm)
then have  $\text{perm-pt.supp } ?q = (\text{perm-pt.supp } (x \rightleftharpoons y) \cup \text{perm-pt.supp } p)$ 
  by (simp add: supp-plus-perm-eq)
then have  $\text{perm-pt.supp } ?q = \text{insert } x Xs \cup ?q \cdot \text{insert } x Xs$ 
  using p2  $\langle x \neq y \rangle$  unfolding q supp-swap
  by auto
ultimately show ?case by blast
qed
qed

```

```

lemma (in permutation-type) finite-atom-set-avoiding:
  fixes Xs ::  $('a::\text{infinite}) \text{ set}$ 
  assumes finite (supp c)
  and finite Xs
  obtains p ::  $('a::\text{infinite}) \text{ perm}$ 
  where  $(p \cdot Xs) \nparallel c$  and  $\text{perm-pt.supp } p = (Xs \cup (p \cdot Xs))$ 

```

```

using assms and atom-set-avoiding-aux [of Xs Xs  $\cup$  supp c]
unfolding fresh-set-def fresh-def by blast

lemma (in permutation-type) finite-atom-set-avoidingD:
  assumes finite (supp c)
  and finite xs
  shows  $\exists p. (p \cdot xs) \# c$ 
  using assms by (elim finite-atom-set-avoiding) auto

lemma (in permutation-type) permute-minus-comp-id [simp]:
  fixes  $\pi :: ('a::infinite) \text{ perm}$ 
  shows  $((\cdot) (- \pi)) \circ ((\cdot) \pi) = (\text{id} :: 'b \Rightarrow 'b)$ 
  by auto

locale finitely-supported = permutation-type +
  assumes finite-supp: finite (supp x)
begin

lemma atom-set-avoiding:
  fixes Xs :: ('a::infinite) set
  assumes finite Xs
  obtains p :: ('a::infinite) perm
  where  $(p \cdot Xs) \# c$  and perm-pt.supp p =  $(Xs \cup (p \cdot Xs))$ 
  using assms and atom-set-avoiding-aux [of Xs Xs  $\cup$  supp c]
  and finite-atom-set-avoiding [OF finite-supp] by blast

lemma atom-set-avoidingD:
  assumes finite xs
  shows  $\exists p. (p \cdot xs) \# c$ 
  using assms and finite-atom-set-avoidingD [OF finite-supp] by blast

lemma supp-eqvt [eqvt]:
  shows  $p \cdot (\text{supp } x) = \text{supp } (p \cdot x)$ 
proof -
  interpret ap: pred-pt permute-atom ..
  interpret asp: pred-pt permute-atom-set ..
  interpret bf: fun-pt permute-bool permute-bool ..
  have *: fun-pt.permute-fun  $(\cdot) (\cdot) p$  Not = Not
  by (simp add: bf.permute-fun-def permute-bool-def)
  show ?thesis
  unfolding supp-def
  unfolding atom-set-pt.Collect-eqvt
  unfolding atom-pred-pt.lambda-eqvt
  unfolding asp.apply-eqvt [of - infinite]
  by (simp add: bf.apply-eqvt * eqvt asp.permute-pred-def permute-eqvt [of p]
    atom-pt.unpermute-def atom-pred-pt.lambda-eqvt
    del: permute-eq-iff bool-pure.permute-pure)
qed

```

```

lemma rename-avoiding:
  assumes finite Xs
  obtains  $p \ t'$  where  $t' = p \cdot t \ Xs \cap \text{supp } t' = \{\}$ 
proof –
  obtain  $p$  where  $1: - p \cdot Xs \cap \text{supp } t = \{\}$ 
    by (metis assms minus-minus atom-set-avoidingD fresh-set-disjoint)
  obtain  $t'$  where  $t' = p \cdot t$  by simp
  moreover then have  $Xs \cap \text{supp } t' = \{\}$ 
    by (metis 1 atom-set-pt.empty-eqvt atom-set-pt.inter-eqvt atom-set-pt.permute-minus-cancel(1)
supp-eqvt)
  ultimately show ?thesis ..
qed

```

We can always rename finitely supported entities apart.

```

lemma supp-fresh-set:
   $\exists p. \text{supp } (p \cdot x) \# y$ 
  using atom-set-avoidingD [OF finite-supp]
  by (simp add: eqvt)

lemma fresh-eqvt [eqvt]:
  fixes  $a :: 'a$ 
  shows  $p \cdot (a \# x) = (p \cdot a) \# (p \cdot x)$ 
  by (simp add: fresh-def eqvt del: bool-pure.permute-pure)

lemma fresh-permute-iff:
  fixes  $a :: 'a$ 
  shows  $(p \cdot a) \# (p \cdot x) \longleftrightarrow a \# x$ 
  by (simp only: fresh-eqvt [symmetric] permute-bool-def)

```

```

lemma fresh-permute-left:
  fixes  $a :: 'a$ 
  shows  $a \# p \cdot x \longleftrightarrow (- p \cdot a) \# x$ 
proof
  assume  $a \# p \cdot x$ 
  then have  $- p \cdot a \# - p \cdot p \cdot x$  by (simp only: fresh-permute-iff)
  then show  $- p \cdot a \# x$  by simp
next
  assume  $- p \cdot a \# x$ 
  then have  $p \cdot - p \cdot a \# p \cdot x$  by (simp only: fresh-permute-iff)
  then show  $a \# p \cdot x$  by simp
qed

```

end

```

context list-pt
begin

```

adhoc-overloading

```

FRESH fresh elt.fresh

lemma supp-Nil [simp]:
  supp [] = {}
  by (simp add: supp-def)

lemma supp-Cons [simp]:
  supp (x # xs) = elt.supp x ∪ supp xs
  by (simp add: elt.supp-def supp-def Collect-imp-eq Collect-neg-eq)

lemma fresh-Nil [simp]:
  a # []
  by (simp add: fresh-def)

lemma fresh-Cons [iff]:
  a # (x # xs) ⟷ a # x ∧ a # xs
  by (simp add: elt.fresh-def fresh-def)

lemma supp-Union [simp]:
  supp xs = (⋃ x ∈ set xs. elt.supp x)
  by (induct xs) (simp del: permute-list-def) +

end

context prod-pt
begin

lemma supp-Un [simp]:
  supp (x, y) = fst.supp x ∪ snd.supp y
  by (auto simp: supp-def fst.supp-def snd.supp-def eqvt)

lemma fst-eqvt [eqvt]:
  π · (fst p) = fst (π · p)
  by (cases p) (simp add: eqvt)

lemma snd-eqvt [eqvt]:
  π · (snd p) = snd (π · p)
  by (cases p) (simp add: eqvt)

end

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) *term* \Rightarrow *pos list*

where

poss-list (Var *x*) = [] |
poss-list (Fun *f ss*) = ([] # concat (map (λ (*i*, *ps*).
 map ((#) *i*) *ps*) (zip [0..ss] (map *poss-list ss*))))

lemma *poss-list-sound* [*simp*]:

set (*poss-list s*) = *poss s*

proof (*induct s*)

case (Fun *f ss*)

let ?*z* = zip [0..ss] (map *poss-list ss*)

have ($\bigcup a \in \text{set } ?z. \text{set } (\text{case-prod } (\lambda i. \text{map } ((\#) i)) a) =$

$\{i \# p \mid i p. i < \text{length } ss \wedge p \in \text{poss } (ss ! i)\}$) (**is** ?*l* = ?*r*)

proof (*rule set-eqI*)

fix *ip*

show (*ip* \in ?*l*) = (*ip* \in ?*r*)

proof

assume *ip* \in ?*l*

from this obtain *ipI* **where**

z: *ipI* \in *set* ?*z* **and**

ip: *ip* \in *set* (*case-prod* ($\lambda i. \text{map } ((\#) i)$) *ipI*)

by *auto*

from *z* **obtain** *i* **where** *i* < length ?*z* **and** *zi*: ?*z* ! *i* = *ipI*

by (*force simp: set-conv-nth*)

with *ip Fun* **show** *ip* \in ?*r* **by** *auto*

next

assume *ip* \in ?*r*

from this obtain *i p* **where** *i* < length *ss* **and** *p* \in *poss* (*ss* ! *i*)

and *ip*: *ip* = *i* # *p* **by** *auto*

with *Fun* **have** *p*: *p* \in *set* (*poss-list* (*ss* ! *i*)) **and** *iz*: *i* < length ?*z* **by** *auto*

from *i* **have** *id*: ?*z* ! *i* = (*i*, *poss-list* (*ss* ! *i*)) (**is** - = ?*ipI*) **by** *auto*

from *iz* **have** ?*z* ! *i* \in *set* ?*z* **by** (*rule nth-mem*)

with *id* **have** *inZ*: ?*ipI* \in *set* ?*z* **by** *auto*

from *p* **have** *i* # *p* \in *set* (*case-prod* ($\lambda i. \text{map } ((\#) i)$) ?*ipI*) **by** *auto*

with *inZ ip* **show** *ip* \in ?*l* **by** *force*

qed

qed

with *Fun* **show** ?*case* **by** *simp*

qed *simp*

declare *poss-list.simps* [*simp del*]

fun *var-poss-list* :: ('f, 'v) *term* \Rightarrow *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 (induct s)
  case (Fun f ss)
  let ?z = zip [0..<length ss] (map var-poss-list ss)
  have (⋃ a∈set ?z. set (case-prod (λi. map ((#) i) a)) =
    (⋃ i<length ss. {i # p | p. p ∈ var-poss (ss ! i)})) (is ?l = ?r)
  proof (rule set-eqI)
    fix ip
    show (ip ∈ ?l) = (ip ∈ ?r)
  proof
    assume ip ∈ ?l
    from this obtain ipI where
      z: ipI ∈ set ?z and
      ip: ip ∈ set (case-prod (λ i. map ((#) i)) ipI)
    by auto
    from z obtain i where i: i < length ?z and zi: ?z ! i = ipI
    by (force simp: set-conv-nth)
    with ip Fun show ip ∈ ?r by auto
  next
    assume ip ∈ ?r
    from this obtain i p where i: i < length ss and p ∈ var-poss (ss ! i)
    and ip: ip = i # p by auto
    with Fun have p: p ∈ set (var-poss-list (ss ! i)) and iz: i < length ?z by
  auto
    from i have id: ?z ! i = (i, var-poss-list (ss ! i)) (is - = ?ipI) by auto
    from iz have ?z ! i ∈ set ?z by (rule nth-mem)
    with id have inZ: ?ipI ∈ set ?z by auto
    from p have i # p ∈ set (case-prod (λ i. map ((#) i)) ?ipI) by auto
    with inZ ip show ip ∈ ?l by force
  qed
qed
with Fun show ?case unfolding var-poss-list.simps by simp
qed simp

lemma length-var-poss-list: length (var-poss-list t) = length (vars-term-list t)
proof (induct t)
  case (Var x)
  then show ?case unfolding var-poss-list.simps vars-term-list.simps by simp
next
  case (Fun f ts)
  let ?xs=map2 (λx. map ((#) x)) [0..<length ts] (map var-poss-list ts)
  let ?ys=map vars-term-list ts
  have l1:length ?xs = length ts

```

```

    by simp
  have l2:length ?ys = length ts
    by simp
  {fix i assume i:i < length ts
    then have (zip [0..

```

```

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
  using assms proof(induct t arbitrary:i)
  case (Var x)
  then have i:i = 0
    unfolding vars-term-list.simps by simp
  then show ?case unfolding i vars-term-list.simps poss-list.simps var-poss.simps
by simp
next
  case (Fun f ts)
  let ?xs=(map vars-term-list ts)
  let ?ys=(map2 (λi. map ((#) i)) [0..

```

```

var-poss-list ts) ! j)
  using l j by presburger
  then show ?thesis
    unfolding var-poss-list.simps concat(2) using j unfolding length-map by
simp
qed
ultimately show ?case
  by presburger
qed

lemma var-poss-list-map-vars-term:
  shows var-poss-list (map-vars-term f t) = var-poss-list t
proof(induct t)
  case (Fun g ts)
  then have IH:map var-poss-list ts = map var-poss-list (map (map-vars-term f)
ts)
    by fastforce
  then show ?case unfolding map-vars-term-eq eval-term.simps IH var-poss-list.simps
    by force
qed simp

lemma distinct-var-poss-list:
  shows distinct (var-poss-list t)
proof(induct t)
  case (Fun f ts)
  let ?xs=(map2 (λi. map ((#) i)) [0..

```

```

{fix x y assume x ∈ set ?xs y ∈ set ?xs x ≠ y
 then obtain i j where i:i < length ?xs x = ?xs!i and j:j < length ?xs y =
 ?xs!j and ij:i ≠ j
   by (metis in-set-idx)
 from i have x:x = map ((#) i) (var-poss-list (ts!i)) by simp
 from j have y:y = map ((#) j) (var-poss-list (ts!j)) by simp
 {fix p q assume p:p ∈ set x and q:q ∈ set y
  from x p obtain p' where p':p = i#p' and p' ∈ set (var-poss-list (ts!i))
   by auto
  from y q obtain q' where q':q = j#q' and q' ∈ set (var-poss-list (ts!j))
   by auto
  from q' p' have p ≠ q by (simp add: ij)
 }
 then have set x ∩ set y = {} by auto
 }note d3=this
 from d1 d2 d3 show ?case unfolding var-poss-list.simps using distinct-concat-iff
 by blast
qed simp

```

12 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]
by (simp add: vars-term-list.simps(1))

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
by (metis assms in-set-idx nth-mem o-apply set-remdups set-rev)

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 ∈ poss t = *in-poss* p t **by** (induct rule: *in-poss.induct*) auto
end

12.1 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* :: '*v* ⇒ ('*f*, '*v*) *term* ⇒ *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* ∈ *vars-term* *t*
by (*induct* *t*) *auto*

lemma *in-vars-term-code*[*code-unfold*]: *x* ∈ *vars-term* *t* = *contains-var-term* *x* *t*
by *simp*
end

lemma *linear-vars-term-list*:

assumes *linear-term* *t*
 shows *length* (*filter* ((=) *x*) (*vars-term-list* *t*)) ≤ 1

using *assms*

proof (*induct* *t*)

case (Var *y*)
 show ?*case* **by** (*auto simp: vars-term-list.simps*)

next

case (Fun *f ss*)

show ?*case*

proof (*rule ccontr*)

assume ¬ ?*thesis*

from *this* [*unfolded vars-term-list.simps*]

have *len*: 2 ≤ *length* (*filter* ((=) *x*) (*concat* (*map vars-term-list ss*))) (**is** - ≤ *length* ?*xs*) **by** *auto*

from *len* **obtain** *y* *ys* **where** ?*xs* = *y* # *ys* **by** (*cases* ?*xs*, *auto*)

from *len* [*unfolded xs*] **obtain** *z* *zs* **where** ?*ys* = *z* # *zs* **by** (*cases* ?*ys*, *auto*)

from ?*xs* [*unfolded ys*] **have** {*y*, *z*} ⊆ *set* ?*xs* **by** *auto*

from *this* [*unfolded set-filter*] **have** *y* = *x* **and** *z* = *x* **by** *auto*

from ?*xs* [*unfolded ys this*] **have** ?*xs* = *x* # *x* # *zs* **by** *auto*

 {

fix *s*

assume *s*: *s* ∈ *set* *ss*

with *Fun*(2) [*unfolded linear-term.simps*]

have *linear-term* *s* **by** *auto*

note *Fun*(1) [*OF* *s this*]

 }

from *this* *Fun*(2) [*unfolded linear-term.simps*] *xs*

show *False*

```

proof (induct ss)
  case Nil then show ?case by simp
next
  case (Cons s ss) note oCons = this
  from Cons(3) have part: is-partition (map vars-term ss) ∧ (∀ s ∈ set ss.
linear-term s) and lin: linear-term s using is-partition-Cons by auto
  from Cons(1)[OF - part] Cons(2)
  have ind: filter ((=) x) (concat (map vars-term-list ss)) ≠ x # x # zs by
auto
  show ?case
  proof (cases filter ((=) x) (vars-term-list s))
    case Nil
    with Cons(4) ind show False by auto
  next
    case (Cons y ys)
    let ?s = filter ((=) x) (vars-term-list s)
    let ?ss = filter ((=) x) (concat (map vars-term-list ss))
    from Cons oCons(4) have ?s = x # ys by auto
    with oCons(2)[of s] have sx: ?s = [x]
      by auto
    with oCons(4) have ssx: ?ss = x # zs by auto
    from sx have x ∈ set ?s by auto
    from this[unfolded set-filter set-vars-term-list]
    have sx: x ∈ vars-term s by auto
    from ssx have x ∈ set ?ss by auto
    from this[unfolded set-filter set-vars-term-list]
    obtain t where tx: x ∈ vars-term t and t: t ∈ set ss by auto
    from t[unfolded set-conv-nth] obtain i where i: i < length ss and
      t: t = ss ! i by auto
    from oCons(3)[THEN conjunct1, unfolded is-partition-def, THEN spec[of -
Suc i], THEN mp, THEN spec[of - 0]] i tx[unfolded t] sx
    show False by auto
  qed
qed
qed
qed

```

```

lemma linear-term-distinct-vars:
  assumes linear-term t
  shows distinct (vars-term-list t)
  using distinct-alt linear-vars-term-list[OF assms] by blast

```

```

lemma distinct-vars-linear-term:
  assumes distinct (vars-term-list t)
  shows linear-term t
  using assms proof (induct t)
    case (Fun f ts)
    {fix t' assume t':t' ∈ set ts
      with Fun(2) have distinct (vars-term-list t')
    }
```

```

    unfolding vars-term-list.simps by (simp add: distinct-concat-iff)
  with Fun(1) t' have linear-term t' by auto
}note IH=this
have is-partition (map (set ∘ vars-term-list) ts)
  using distinct-is-partition-sets vars-term-list.simps(2) Fun(2) by force
then have is-partition (map vars-term ts) by (simp add: comp-def)
with IH show ?case by simp
qed simp

```

```

fun supseq-list :: ('f, 'v) term ⇒ ('f, 'v) term list
where
  supseq-list (Var x) = [Var x] |
  supseq-list (Fun f ts) = Fun f ts # concat (map supseq-list ts)

```

```

fun supt-list :: ('f, 'v) term ⇒ ('f, 'v) term list
where
  supt-list (Var x) = [] |
  supt-list (Fun f ts) = concat (map supseq-list ts)

```

```

lemma supseq-list [simp]:
  set (supseq-list t) = {s. t ⊇ s}
proof (rule set-eqI, unfold mem-Collect-eq)
  fix s
  show s ∈ set(supseq-list t) = (t ⊇ s)
  proof (induct t)
    case (Fun f ss)
    show ?case
    proof (cases Fun f ss = s)
      case False
      show ?thesis
    proof
      assume Fun f ss ⊇ s
      with False have sup: Fun f ss ⊃ s using supseq-supt-conv by auto
      obtain C where C ≠ □ and Fun f ss = C⟨s⟩ using sup by auto
      then obtain b D a where Fun f ss = Fun f (b @ D⟨s⟩ # a) by (cases C,
auto)
      then have D: D⟨s⟩ ∈ set ss by auto
      with Fun[OF D] ctxt-imp-supseq[of D s] obtain t where t ∈ set ss and s
∈ set (supseq-list t) by auto
      then show s ∈ set (supseq-list (Fun f ss)) by auto
    next
      assume s ∈ set (supseq-list (Fun f ss))
      with False obtain t where t: t ∈ set ss and s ∈ set (supseq-list t) by auto
      with Fun[OF t] have t ⊇ s by auto
      with t show Fun f ss ⊇ s by auto
    qed
  qed auto
qed (simp add: supseq-var-imp-eq)

```

qed

lemma *supt-list-sound* [*simp*]:

$set (supt-list\ t) = \{s. t \triangleright s\}$

by (*cases t*) *auto*

fun *supt-impl* :: (*'f*, *'v*) *term* \Rightarrow (*'f*, *'v*) *term* \Rightarrow *bool*

where

supt-impl (*Var x*) *t* \longleftrightarrow *False* |

supt-impl (*Fun f ss*) *t* $\longleftrightarrow t \in set\ ss \vee Bex\ (set\ ss)\ (\lambda s. supt-impl\ s\ t)$

lemma *supt-impl* [*code-unfold*]:

$s \triangleright t \longleftrightarrow supt-impl\ s\ t$

proof

assume $s \triangleright t$ **then show** *supt-impl s t*

proof (*induct s*)

case (*Var x*) **then show** *?case* **by** *auto*

next

case (*Fun f ss*) **then show** *?case*

proof (*cases t* $\in set\ ss$)

case *True* **then show** *?thesis* **by** (*simp*)

next

case *False*

assume $\bigwedge s. [s \in set\ ss; s \triangleright t] \implies supt-impl\ s\ t$

and $Fun\ f\ ss \triangleright t$ **and** $t \notin set\ ss$

moreover from $\langle Fun\ f\ ss \triangleright t \rangle$ **obtain** *s* **where** $s \in set\ ss$ **and** $s \triangleright t$

by (*cases rule: supt.cases*) (*simp-all add: $\langle t \notin set\ ss \rangle$*)

ultimately have *supt-impl s t* **by** *simp*

with $\langle s \in set\ ss \rangle$ **show** *?thesis* **by** *auto*

qed

qed

next

assume *supt-impl s t*

then show $s \triangleright t$

proof (*induct s*)

case (*Var x*) **then show** *?case* **by** *simp*

next

case (*Fun f ss*)

then have $t \in set\ ss \vee (\exists s \in set\ ss. supt-impl\ s\ t)$ **by** *simp*

then show *?case*

proof

assume $t \in set\ ss$ **then show** *?case* **by** *auto*

next

assume $\exists s \in set\ ss. supt-impl\ s\ t$

then obtain *s* **where** $s \in set\ ss$ **and** *supt-impl s t* **by** *auto*

with *Fun* **have** $s \triangleright t$ **by** *auto*

with $\langle s \in set\ ss \rangle$ **show** *?thesis* **by** *auto*

qed

qed

qed

lemma *supteq-impl*[code-unfold]: $s \supseteq t \iff s = t \vee \text{supt-impl } s \ t$
unfolding *supteq-supt-set-conv*
by (*auto simp: supt-impl*)

fun

linear-term-impl :: $'v \text{ set} \Rightarrow ('f, 'v) \text{ term} \Rightarrow ('v \text{ set}) \text{ option}$

where

linear-term-impl *xs* (*Var* *x*) = (if $x \in xs$ then *None* else *Some* (*insert* *x* *xs*)) |
linear-term-impl *xs* (*Fun* - []) = *Some* *xs* |
linear-term-impl *xs* (*Fun* *f* (*t* # *ts*)) = (case *linear-term-impl* *xs* *t* of
None \Rightarrow *None*
| *Some* *ys* \Rightarrow *linear-term-impl* *ys* (*Fun* *f* *ts*))

lemma *linear-term-code*[code]: *linear-term* *t* = (*linear-term-impl* {} *t* \neq *None*)

proof -

{
note [*simp*] = *is-partition-Nil is-partition-Cons*
fix *xs* *ys*
let $?P = \lambda \text{ys xs t. } (\text{linear-term-impl xs t} = \text{None} \longrightarrow (xs \cap \text{vars-term t} \neq \{\}) \vee \neg \text{linear-term t}) \wedge$
 $(\text{linear-term-impl xs t} = \text{Some ys} \longrightarrow (ys = (xs \cup \text{vars-term t})) \wedge xs \cap \text{vars-term t} = \{\}) \wedge \text{linear-term t})$
have $?P \text{ys xs t}$
proof (*induct* rule: *linear-term-impl.induct*[of $\lambda \text{xs t. } \forall \text{ys. } ?P \text{ys xs t, rule-format}$])
case ($\exists \text{xs f t ts zs}$)
show $?case$
proof (*cases* *linear-term-impl* *xs* *t*)
case *None*
with \exists **show** $?thesis$ **by** *auto*
next
case (*Some* *ys*)
note *some* = *this*
with \exists **have** *rec1*: $ys = xs \cup \text{vars-term t} \wedge xs \cap \text{vars-term t} = \{\} \wedge$
 linear-term t **by** *auto*
show $?thesis$
proof (*cases* *linear-term-impl* *ys* (*Fun* *f* *ts*))
case *None*
with *rec1* *Some* **have** *res*: *linear-term-impl* *xs* (*Fun* *f* (*t* # *ts*)) = *None*
by *simp*
from *None* $\exists(2)$ *Some* **have** *rec2*: $ys \cap \text{vars-term} (\text{Fun } f \text{ ts}) \neq \{\} \vee \neg$
 $\text{linear-term} (\text{Fun } f \text{ ts})$ **by** *simp*
then **have** $xs \cap \text{vars-term} (\text{Fun } f (\text{t} \# \text{ts})) \neq \{\} \vee \neg \text{linear-term} (\text{Fun } f$
 $(\text{t} \# \text{ts}))$
proof
assume $ys \cap \text{vars-term} (\text{Fun } f \text{ ts}) \neq \{\}$
then **obtain** *x* **where** *x1*: $x \in \text{vars-term} (\text{Fun } f \text{ ts})$ **by**
auto

```

    show ?thesis
  proof (cases  $x \in xs$ )
    case True
      with  $x2$  show ?thesis by auto
    next
      case False
        with  $x1$  rec1 have  $x \in \text{vars-term } t$  by auto
        with  $x2$  have  $\neg \text{linear-term } (\text{Fun } f \ (t \# \ ts))$  by auto
        then show ?thesis ..
      qed
    next
      assume  $\neg \text{linear-term } (\text{Fun } f \ ts)$  then have  $\neg \text{linear-term } (\text{Fun } f \ (t \# \ ts))$  by auto
      then show ?thesis ..
    qed
  with res show ?thesis by auto
next
case (Some us)
with some have res:  $\text{linear-term-impl } xs \ (\text{Fun } f \ (t \# \ ts)) = \text{Some } us$  by
auto
{
  assume us:  $us = zs$ 
  from Some[simplified us] 3(2) some
  have rec2:  $zs = ys \cup \text{vars-term } (\text{Fun } f \ ts) \wedge ys \cap \text{vars-term } (\text{Fun } f \ ts)$ 
=  $\{\} \wedge \text{linear-term } (\text{Fun } f \ ts)$  by auto
  from rec1 rec2
  have part1:  $zs = xs \cup \text{vars-term } (\text{Fun } f \ (t \# \ ts)) \wedge xs \cap \text{vars-term } (\text{Fun } f \ (t \# \ ts)) = \{\}$  (is ?part1) by auto
  from rec1 rec2 have  $\text{vars-term } t \cap \text{vars-term } (\text{Fun } f \ ts) = \{\}$  and
 $\text{linear-term } t$  and  $\text{linear-term } (\text{Fun } f \ ts)$  by auto
  then have  $\text{linear-term } (\text{Fun } f \ (t \# \ ts))$  (is ?part2) by auto
  with part1 have ?part1  $\wedge$  ?part2 ..
}
with res show ?thesis by auto
qed
qed
qed auto
} note main = this
from main[of  $\{\}$ ] show ?thesis by (cases  $\text{linear-term-impl } \{\} \ t, \text{ auto}$ )
qed

```

definition $\text{check-no-var} :: ('f::\text{showl}, 'v::\text{showl}) \text{ term} \Rightarrow \text{showsl check}$ **where**
 $\text{check-no-var } t \equiv \text{check } (\text{is-Fun } t) \ (\text{showsl } (\text{STR } \text{"variable found"} \ \boxed{\leftarrow}))$

lemma $\text{check-no-var-sound[simp]}$:
 $\text{isOK } (\text{check-no-var } t) \longleftrightarrow \text{is-Fun } t$
unfolding check-no-var-def **by** simp

definition

$check_supt :: ('f::showl, 'v::showl) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow showsl \text{ check}$

where

$check_supt \ s \ t \equiv check \ (s \triangleright t) \ (showsl \ t \circ showsl \ (STR \ '' \text{ is not a proper subterm of } '') \circ showsl \ s)$

definition

$check_supteq :: ('f::showl, 'v::showl) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow showsl \text{ check}$

where

$check_supteq \ s \ t \equiv check \ (s \supseteq t) \ (showsl \ t \circ showsl \ (STR \ '' \text{ is not a subterm of } '') \circ showsl \ s)$

lemma *isOK-check-supt* [simp]:

$isOK \ (check_supt \ s \ t) \longleftrightarrow s \triangleright t$

by (auto simp: check-supt-def)

lemma *isOK-check-supteq* [simp]:

$isOK \ (check_supteq \ s \ t) \longleftrightarrow s \supseteq t$

by (auto simp: check-supteq-def)

12.2 Additional Functions on Terms

fun *with-arity* :: ('f, 'v) term \Rightarrow ('f \times nat, 'v) term **where**

$with_arity \ (Var \ x) = Var \ x$

| $with_arity \ (Fun \ f \ ts) = Fun \ (f, length \ ts) \ (map \ with_arity \ ts)$

fun *add-vars-term* :: ('f, 'v) term \Rightarrow 'v list \Rightarrow 'v list**where**

$add_vars_term \ (Var \ x) \ xs = x \# \ xs$ |

$add_vars_term \ (Fun \ f \ ts) \ xs = foldr \ add_vars_term \ ts \ xs$

fun *add-funs-term* :: ('f, 'v) term \Rightarrow 'f list \Rightarrow 'f list**where**

$add_funs_term \ (Var \ -) \ fs = fs$ |

$add_funs_term \ (Fun \ f \ ts) \ fs = f \# foldr \ add_funs_term \ ts \ fs$

fun *add-funas-term* :: ('f, 'v) term \Rightarrow ('f \times nat) list \Rightarrow ('f \times nat) list**where**

$add_funas_term \ (Var \ -) \ fs = fs$ |

$add_funas_term \ (Fun \ f \ ts) \ fs = (f, length \ ts) \# foldr \ add_funas_term \ ts \ fs$

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

$add_funas_args_term \ t \ fs = foldr \ add_funas_term \ (args \ t) \ fs$

lemma *add-vars-term-vars-term-list-conv* [simp]:

$add_vars_term \ t \ xs = vars_term_list \ t \ @ \ xs$

proof (induct t arbitrary: xs)

case (Fun f ts)

then show *?case by (induct ts) (simp-all add: vars-term-list.simps)*
qed (*simp add: vars-term-list.simps*)

lemma *add-funs-term-funs-term-list-conv [simp]:*
add-funs-term t fs = funs-term-list t @ fs
proof (*induct t arbitrary: fs*)
case (*Fun f ts*)
then show *?case by (induct ts) (simp-all add: funs-term-list.simps)*
qed (*simp add: funs-term-list.simps*)

lemma *add-funas-term-funas-term-list-conv [simp]:*
add-funas-term t fs = funas-term-list t @ fs
proof (*induct t arbitrary: fs*)
case (*Fun f ts*)
then show *?case by (induct ts) (simp-all add: funas-term-list.simps)*
qed (*simp add: funas-term-list.simps*)

lemma *add-vars-term-vars-term-list-abs-conv [simp]:*
add-vars-term = (@) ∘ vars-term-list
by (*intro ext simp*)

lemma *add-funs-term-funs-term-list-abs-conv [simp]:*
add-funs-term = (@) ∘ funs-term-list
by (*intro ext simp*)

lemma *add-funas-term-funas-term-list-abs-conv [simp]:*
add-funas-term = (@) ∘ funas-term-list
by (*intro ext simp*)

lemma *add-funas-args-term-funas-args-term-list-conv [simp]:*
add-funas-args-term t fs = funas-args-term-list t @ fs
by (*simp add: add-funas-args-term-def funas-args-term-list-def concat-conv-foldr foldr-map*)

fun *insert-vars-term :: ('f, 'v) term ⇒ 'v list ⇒ '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 ⇒ 'f list ⇒ '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 ⇒ ('f × nat) list ⇒ ('f × 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

proof (*induct t arbitrary: xs*)

case (*Fun f ts*)

then show ?*case* **by** (*induct ts*) *auto*

qed *simp*

lemma *set-insert-funs-term-funs-term* [*simp*]:

set (insert-funs-term t fs) = funs-term t \cup set fs

proof (*induct t arbitrary: fs*)

case (*Fun f ts*)

then show ?*case* **by** (*induct ts*) *auto*

qed *simp*

lemma *set-insert-funas-term-funas-term* [*simp*]:

set (insert-funas-term t fs) = funas-term t \cup set fs

proof (*induct t arbitrary: fs*)

case (*Fun f ts*)

then have *set (foldr insert-funas-term ts fs) = \bigcup (funas-term ‘ set ts) \cup set fs*
by (*induct ts*) *auto*

then show ?*case* **by** *simp*

qed *simp*

lemma *set-insert-funas-args-term* [*simp*]:

set (insert-funas-args-term t fs) = \bigcup (funas-term ‘ set (args t)) \cup set fs

proof (*induct t arbitrary: fs*)

case (*Fun f ts*)

then show ?*case* **by** (*induct ts*) (*auto simp: insert-funas-args-term-def*)

qed (*simp add: insert-funas-args-term-def*)

Implementations of corresponding set-based functions.

abbreviation *vars-term-impl* *t* \equiv *insert-vars-term* *t* []

abbreviation *funs-term-impl* *t* \equiv *insert-funs-term* *t* []

abbreviation *funas-term-impl* *t* \equiv *insert-funas-term* *t* []

lemma [*code*]:

vars-term-list *t* = *add-vars-term* *t* []

funs-term-list *t* = *add-funs-term* *t* []

by *simp-all*

lemma *with-arity-term-fold* [*code*]:

```

    with-arity = Term-More.fold Var ( $\lambda f\ ts.$  Fun ( $f,$  length  $ts$ )  $ts$ )
proof
  fix  $t :: ('f, 'v)$  term
  show with-arity  $t =$  Term-More.fold Var ( $\lambda f\ ts.$  Fun ( $f,$  length  $ts$ )  $ts$ )  $t$ 
    by (induct  $t$ ) simp-all
qed

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 (induct  $t$ )
  case (Var  $x$ )
  show ?case (is - = ? $R$ )
  proof -
    {
      fix  $t$ 
      assume  $t \in ?R$ 
      then have  $t =$  Var  $x$  by (cases  $t$ , auto)
    }
    then show ?thesis by auto
  qed
next
  case (Fun  $fs\ ts$ )
  show ?case (is ? $L = ?R$ )
  proof -
    {
      fix  $i$ 
      assume  $i < \text{length } ts$ 
      then have  $ts ! i \in \text{set } ts$  by auto
      note Fun[OF this]
    } note ind = this
    have idL: ? $L =$  {Fun  $g\ ss \mid g\ ss. g \in \text{set } fs \wedge \text{length } ss = \text{length } ts \wedge (\forall i < \text{length } ts. ss ! i \in \text{set } (\text{flatten-term-enum } (ts ! i)))$ } (is - = ? $M1$ ) by auto
    let ? $R1 =$  {Fun  $f\ ss \mid f\ ss. f \in \text{set } fs \wedge \text{length } ss = \text{length } ts \wedge (\forall i < \text{length } ss. \text{instance-term } (ss ! i) (\text{map-funs-term set } (ts ! i)))$ }
    {
      fix  $u$ 
      assume  $u \in ?R$ 
      then have  $u \in ?R1$  by (cases  $u$ , auto)
    }
    then have idR: ? $R = ?R1$  by auto

```

show ?case unfolding idL idR using ind by auto
qed
qed

definition *mk-subst-domain* :: ('f, 'v) substL \Rightarrow ('v \times ('f, 'v) term) list **where**
mk-subst-domain $\sigma \equiv$
let $\tau = \text{mk-subst Var } \sigma$ in
(filter ($\lambda(x, t). \text{Var } x \neq t$) (map ($\lambda x. (x, \tau x)$) (remdups (map fst σ))))

lemma *mk-subst-domain*:

set (*mk-subst-domain* σ) = ($\lambda x. (x, \text{mk-subst Var } \sigma x)$) ' *subst-domain* (*mk-subst*
Var σ)
(is ?I = ?R)

proof –

have ?I \subseteq ?R unfolding *mk-subst-domain-def* *Let-def* *subst-domain-def* by auto
moreover

{
fix xt
assume mem: $xt \in ?R$
obtain x t **where** $xt: xt = (x, t)$ **by** force
from mem [unfolded xt]
have $x: x \in \text{subst-domain} (\text{mk-subst Var } \sigma)$ **and** $t: t = \text{mk-subst Var } \sigma x$ **by**
auto
then have $\text{mk-subst Var } \sigma x \neq \text{Var } x$ unfolding *subst-domain-def* **by** simp
with t have l: map-of $\sigma x = \text{Some } t$ **and** tx: $t \neq \text{Var } x$
unfolding *mk-subst-def* **by** (cases map-of σx , auto)
from map-of-SomeD[OF l] l t tx have $(x, t) \in ?I$ unfolding *mk-subst-domain-def*
Let-def
by force
then have $xt \in ?I$ unfolding xt .
}
ultimately show ?thesis **by** blast
qed

lemma *finite-mk-subst*: finite (*subst-domain* (*mk-subst* Var σ))

proof –

have *subst-domain* (*mk-subst* Var σ) = fst ' set (*mk-subst-domain* σ)
unfolding *mk-subst-domain* *Let-def* **by** force
moreover have finite ...
using *finite-set* **by** auto
ultimately show ?thesis **by** simp
qed

definition *subst-eq* :: ('f, 'v) substL \Rightarrow ('f, 'v) substL \Rightarrow bool **where**

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*]:

subst-eq $\sigma \tau = (\text{mk-subst Var } \sigma = \text{mk-subst Var } \tau)$

```

proof –
  let ?σ = mk-subst Var σ
  let ?τ = mk-subst Var τ
  {
    assume id: ((λ x. (x, ?σ x)) ‘ subst-domain ?σ) = ((λ x. (x, ?τ x)) ‘
subst-domain ?τ) (is ?l = ?r)
    from arg-cong[OF id, of (‘) fst] have idd: subst-domain ?σ = subst-domain ?τ
by force
    have ?σ = ?τ
    proof (rule ext)
      fix x
      show ?σ x = ?τ x
      proof (cases x ∈ subst-domain ?σ)
        case False
          then show ?thesis using idd unfolding subst-domain-def by auto
        next
          case True
            with idd have x: (x, ?σ x) ∈ ?l (x, ?τ x) ∈ ?r by auto
            with id have x: (x, ?τ x) ∈ ?l (x, ?σ x) ∈ ?l by auto
            then show ?thesis by auto
          qed
        qed
      }
    then show ?thesis
      unfolding subst-eq-def Let-def
      unfolding mk-subst-domain by auto
    qed
  }

```

definition range-vars-impl :: ('f, 'v) substL ⇒ 'v list

where

```

range-vars-impl σ =
  (let σ' = mk-subst-domain σ in
   concat (map (vars-term-list o snd) σ'))

```

definition vars-subst-impl :: ('f, 'v) substL ⇒ 'v list

where

```

vars-subst-impl σ =
  (let σ' = mk-subst-domain σ in
   map fst σ' @ concat (map (vars-term-list o snd) σ'))

```

lemma vars-subst-impl [simp]:

```

set (vars-subst-impl σ) = vars-subst (mk-subst Var σ)
unfolding vars-subst-def vars-subst-impl-def Let-def
by (auto simp: mk-subst-domain, force)

```

lemma range-vars-impl [simp]:

```

set (range-vars-impl σ) = range-vars (mk-subst Var σ)
unfolding range-vars-def range-vars-impl-def Let-def
by (auto simp: mk-subst-domain)

```

lemma *mk-subst-one* [simp]: *mk-subst* Var [(*x*, *t*)] = *subst* *x* *t*
unfolding *mk-subst-def* *subst-def* **by** *auto*

lemma *fst-image* [simp]: *fst* ‘(λ *x*. (*x*, *g* *x*)) ‘ *a* = *a* **by** *force*

definition

subst-compose-impl :: (*f*, *v*) *substL* \Rightarrow (*f*, *v*) *substL* \Rightarrow (*f*, *v*) *substL*

where

subst-compose-impl σ $\tau \equiv$

let

$\sigma' = \text{mk-subst-domain } \sigma;$

$\tau' = \text{mk-subst-domain } \tau;$

$d\sigma = \text{map fst } \sigma'$

in map (λ (*x*, *t*). (*x*, *t* \cdot *mk-subst* Var τ')) $\sigma' @ \text{filter } (\lambda$ (*x*, *t*). *x* \notin *set* *d* σ) τ'

lemma *mk-subst-mk-subst-domain* [simp]:

mk-subst Var (*mk-subst-domain* σ) = *mk-subst* Var σ

proof (*intro ext*)

fix *x*

{

assume *x*: *x* \notin *subst-domain* (*mk-subst* Var σ)

then have σ : *mk-subst* Var σ *x* = Var *x* **unfolding** *subst-domain-def* **by** *auto*

from *x* **have** *x* \notin *fst* ‘*set* (*mk-subst-domain* σ) **unfolding** *mk-subst-domain*

by *auto*

then have *look*: *map-of* (*mk-subst-domain* σ) *x* = *None* **by** (*cases map-of* (*mk-subst-domain* σ) *x*, *insert map-of-SomeD*[*of mk-subst-domain* σ *x*], *force*+))

then have *mk-subst* Var (*mk-subst-domain* σ) *x* = *mk-subst* Var σ *x* **unfolding**

σ

unfolding *mk-subst-def* **by** *auto*

} **note** *ndom* = *this*

{

assume *x* \in *subst-domain* (*mk-subst* Var σ)

then have *x* \in *fst* ‘*set* (*mk-subst-domain* σ) **unfolding** *mk-subst-domain* **by**

auto

then obtain *t* **where** *look*: *map-of* (*mk-subst-domain* σ) *x* = *Some* *t* **by** (*cases map-of* (*mk-subst-domain* σ) *x*, (*force simp: map-of-eq-None-iff*)+))

from *map-of-SomeD*[*OF look*, *unfolded mk-subst-domain*] **have** *t*: *t* = *mk-subst* Var σ *x* **by** *auto*

from *look* *t* **have** *res*: *mk-subst* Var (*mk-subst-domain* σ) *x* = *mk-subst* Var σ *x* **unfolding** *mk-subst-def* **by** *auto*

} **note** *dom* = *this*

from *ndom* *dom*

show *mk-subst* Var (*mk-subst-domain* σ) *x* = *mk-subst* Var σ *x* **by** *auto*

qed

lemma *subst-compose-impl* [simp]:

mk-subst Var (*subst-compose-impl* σ τ) = *mk-subst* Var $\sigma \circ_s \text{mk-subst Var } \tau$ (*is ?l = ?r*)

```

proof (rule ext)
  fix x
  let ?σ = mk-subst Var σ
  let ?τ = mk-subst Var τ
  let ?s = map (λ (x, t). (x, t · mk-subst Var (mk-subst-domain τ))) (mk-subst-domain
σ)
  let ?t = [(x, t) ← mk-subst-domain τ. x ∉ set (map fst (mk-subst-domain σ))]
  note d = subst-compose-impl-def[unfolded Let-def]
  show ?l x = ?r x
  proof (cases x ∈ subst-domain (mk-subst Var σ))
    case True
      then have ?σ x ≠ Var x unfolding subst-domain-def by auto
      then obtain t where look: map-of σ x = Some t and σ: ?σ x = t
        unfolding mk-subst-def by (cases map-of σ x, auto)
      from σ have r: ?r x = t · ?τ unfolding subst-compose-def by simp
      from True have x ∈ subst-domain (mk-subst Var (mk-subst-domain σ))
        by simp
      from σ True have mem: (x, t · ?τ) ∈ set ?s by (auto simp: mk-subst-domain)
      with map-of-eq-None-iff[of ?s x]
        obtain u where look2: map-of ?s x = Some u
        by (cases map-of ?s x, force+)
      from map-of-SomeD[OF this] σ have u: u = t · ?τ
        by (auto simp: mk-subst-domain)
      note look2 = map-of-append-Some[OF look2, of ?t]
      have l: ?l x = t · ?τ unfolding d mk-subst-def[of Var ?s @ ?t] look2 u
        by simp
      from l r show ?thesis by simp
    next
      case False
      then have σ: ?σ x = Var x unfolding subst-domain-def by auto
      from σ have r: ?r x = ?τ x unfolding subst-compose-def by simp
      from False have x ∉ subst-domain (mk-subst Var (mk-subst-domain σ))
        by simp
      from False have mem: ∧ y. (x, y) ∉ set ?s by (auto simp: mk-subst-domain)
      with map-of-SomeD[of ?s x] have look2: map-of ?s x = None
        by (cases map-of ?s x, auto)
      note look2 = map-of-append-None[OF look2, of ?t]
      have l: ?l x = (case map-of ?t x of None ⇒ Var x | Some t ⇒ t) unfolding d
mk-subst-def[of Var ?s @ ?t] look2 by simp
      also have ... = ?τ x
      proof (cases x ∈ subst-domain ?τ)
        case True
          then have ?τ x ≠ Var x unfolding subst-domain-def by auto
          then obtain t where look: map-of τ x = Some t and τ: ?τ x = t
            unfolding mk-subst-def by (cases map-of τ x, auto)
          from True have x ∈ subst-domain (mk-subst Var (mk-subst-domain τ))
            by simp
          from τ True have mem: (x, ?τ x) ∈ set ?t using False by (auto simp:
mk-subst-domain)

```

```

with map-of-eq-None-iff[of ?t x] obtain u where look2: map-of ?t x = Some
u
  by (cases map-of ?t x, force+)
from map-of-SomeD[OF this]  $\tau$  have u: u = ? $\tau$  x
  by (auto simp: mk-subst-domain)
show ?thesis using look2 u by simp
next
  case False
  then have  $\tau$ : ? $\tau$  x = Var x unfolding subst-domain-def by auto
  from False have x  $\notin$  subst-domain (mk-subst Var (mk-subst-domain  $\tau$ ))
    by simp
  from False have mem:  $\bigwedge y. (x, y) \notin \text{set } ?t$  by (auto simp: mk-subst-domain)
  with map-of-SomeD[of ?t x] have look2: map-of ?t x = None
    by (cases map-of ?t x, auto)
  show ?thesis unfolding  $\tau$  look2 by simp
qed
finally show ?thesis unfolding r by simp
qed
qed

fun subst-power-impl :: (f, v) substL  $\Rightarrow$  nat  $\Rightarrow$  (f, v) substL where
  subst-power-impl  $\sigma$  0 = []
| subst-power-impl  $\sigma$  (Suc n) = subst-compose-impl  $\sigma$  (subst-power-impl  $\sigma$  n)

lemma subst-power-impl [simp]:
  mk-subst Var (subst-power-impl  $\sigma$  n) = (mk-subst Var  $\sigma$ )  $\sim^n$ 
  by (induct n, auto)

definition commutes-impl :: (f, v) substL  $\Rightarrow$  (f, v) substL  $\Rightarrow$  bool where
  commutes-impl  $\sigma$   $\mu$   $\equiv$  subst-eq (subst-compose-impl  $\sigma$   $\mu$ ) (subst-compose-impl  $\mu$ 
 $\sigma$ )

lemma commutes-impl [simp]:
  commutes-impl  $\sigma$   $\mu$  = ((mk-subst Var  $\sigma$   $\circ_s$  mk-subst Var  $\mu$ ) = (mk-subst Var  $\mu$ 
 $\circ_s$  mk-subst Var  $\sigma$ ))
  unfolding commutes-impl-def by simp

definition
  subst-compose'-impl :: (f, v) substL  $\Rightarrow$  (f, v) subst  $\Rightarrow$  (f, v) substL
where
  subst-compose'-impl  $\sigma$   $\varrho$   $\equiv$  map ( $\lambda (x, s). (x, s \cdot \varrho)$ ) (mk-subst-domain  $\sigma$ )

lemma subst-compose'-impl [simp]:
  mk-subst Var (subst-compose'-impl  $\sigma$   $\varrho$ ) = subst-compose' (mk-subst Var  $\sigma$ )  $\varrho$  (is
?l = ?r)
proof (rule ext)
  fix x
  note d = subst-compose'-def subst-compose'-impl-def
  let ? $\sigma$  = mk-subst Var  $\sigma$ 

```

```

let ?s = subst-compose'-impl  $\sigma$   $\varrho$ 
show ?l x = ?r x
proof (cases x  $\in$  subst-domain (mk-subst Var  $\sigma$ ))
  case True
  then have r: ?r x = ? $\sigma$  x  $\cdot$   $\varrho$  unfolding d by simp
  from True have (x, ? $\sigma$  x)  $\in$  set (mk-subst-domain  $\sigma$ ) unfolding mk-subst-domain
by auto
  then have (x, ? $\sigma$  x  $\cdot$   $\varrho$ )  $\in$  set ?s unfolding d by auto
  with map-of-eq-None-iff[of ?s x] obtain u where look: map-of ?s x = Some u
  by (cases map-of ?s x, force+)
  from map-of-SomeD[OF this] have u: u = ? $\sigma$  x  $\cdot$   $\varrho$  unfolding d using
mk-subst-domain[of  $\sigma$ ] by auto
  then have l: ?l x = ? $\sigma$  x  $\cdot$   $\varrho$  using look u unfolding mk-subst-def by auto
  from l r show ?thesis by simp
next
  case False
  then have r: ?r x = Var x unfolding d by simp
  from False have  $\bigwedge y. (x, y) \notin$  set ?s unfolding d
  by (auto simp: mk-subst-domain)
  with map-of-SomeD[of ?s x] have look: map-of ?s x = None
  by (cases map-of ?s x, auto)
  then have l: ?l x = Var x unfolding mk-subst-def by simp
  from l r show ?thesis by simp
qed
qed

```

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 ($\lambda (y, t). y \neq x$) σ

lemma *subst-replace-impl* [simp]:

mk-subst Var (*subst-replace-impl* σ x t) = ($\lambda y. \text{if } x = y \text{ then } t \text{ else } \text{mk-subst Var } \sigma y$) (is ?l = ?r)

proof (rule ext)

fix y

note d = *subst-replace-impl-def*

show ?l y = ?r y

proof (cases y = x)

case True

then show ?thesis unfolding d *mk-subst-def* by auto

next

case False

let ? σ = *mk-subst* Var σ

from False have r: ?r y = ? σ y by auto

from False have l: ?l y = *mk-subst* Var ((y, t) \leftarrow σ . y \neq x) y unfolding *mk-subst-def* d

by simp

also have ... = ? σ y unfolding *mk-subst-def*

```

    using map-of-filter[of  $\lambda y. y \neq x y \sigma$ , OF False] by simp
  finally show ?thesis using r by simp
qed
qed

```

```

lemma mk-subst-domain-distinct: distinct (map fst (mk-subst-domain  $\sigma$ ))
  unfolding mk-subst-domain-def Let-def distinct-map
  by (rule conjI[OF distinct-filter], auto simp: distinct-map inj-on-def)

```

```

definition is-renaming-impl :: ( $f, v$ ) substL  $\Rightarrow$  bool where
  is-renaming-impl  $\sigma \equiv$ 
    let  $\sigma' = \text{map snd (mk-subst-domain } \sigma)$  in
    ( $\forall t \in \text{set } \sigma'. \text{is-Var } t \wedge \text{distinct } \sigma'$ )

```

```

lemma is-renaming-impl [simp]:

```

```

  is-renaming-impl  $\sigma = \text{is-renaming (mk-subst Var } \sigma)$  (is ?l = ?r)

```

```

proof -

```

```

  let ? $\sigma = \text{mk-subst Var } \sigma$ 

```

```

  let ? $d = \text{mk-subst-domain } \sigma$ 

```

```

  let ? $m = \text{map snd } ?d$ 

```

```

  let ? $k = \text{map fst } ?d$ 

```

```

  have ?l = ( $\forall t \in \text{set } ?m. \text{is-Var } t \wedge \text{distinct } ?m$ ) unfolding is-renaming-impl-def

```

```

  Let-def by auto

```

```

  also have ( $\forall t \in \text{set } ?m. \text{is-Var } t$ ) = ( $\forall x. \text{is-Var } (? \sigma x)$ )

```

```

    by (force simp: mk-subst-domain subst-domain-def)

```

```

  also have distinct ? $m = \text{inj-on } ? \sigma (\text{subst-domain } ? \sigma)$ 

```

```

  proof

```

```

    assume inj: inj-on ? $\sigma$  (subst-domain ? $\sigma$ )

```

```

    show distinct ? $m$  unfolding distinct-conv-nth

```

```

    proof (intro allI impI)

```

```

      fix i j

```

```

      assume i:  $i < \text{length } ?m$  and j:  $j < \text{length } ?m$  and ij:  $i \neq j$ 

```

```

      obtain x t where di: ? $d ! i = (x, t)$  by (cases ? $d ! i$ , auto)

```

```

      obtain y s where dj: ? $d ! j = (y, s)$  by (cases ? $d ! j$ , auto)

```

```

      from di i have mi: ? $m ! i = t$  and ki: ? $k ! i = x$  by auto

```

```

      from dj j have mj: ? $m ! j = s$  and kj: ? $k ! j = y$  by auto

```

```

      from di i have xt:  $(x, t) \in \text{set } ?d$  unfolding set-conv-nth by force

```

```

      from dj j have ys:  $(y, s) \in \text{set } ?d$  unfolding set-conv-nth by force

```

```

      from xt ys have d:  $x \in \text{subst-domain } ? \sigma \wedge y \in \text{subst-domain } ? \sigma$  unfolding

```

```

      mk-subst-domain by auto

```

```

      have dist: distinct ? $k$  by (rule mk-subst-domain-distinct)

```

```

      from ij i j have xy:  $x \neq y$  unfolding ki[symmetric] kj[symmetric]

```

```

        using dist[unfolded distinct-conv-nth] by auto

```

```

      from xt ys have m: ? $\sigma x = t \wedge ? \sigma y = s$  unfolding mk-subst-domain by auto

```

```

      from inj[unfolded inj-on-def, rule-format, OF d]

```

```

      show ? $m ! i \neq ?m ! j$  unfolding m mi mj using xy by auto

```

```

    qed

```

```

  next

```

```

assume dist: distinct ?m
show inj-on ? $\sigma$  (subst-domain ? $\sigma$ ) unfolding inj-on-def
proof (intro ballI impI)
  fix x y
  assume x: x  $\in$  subst-domain ? $\sigma$  and y: y  $\in$  subst-domain ? $\sigma$ 
  and id: ? $\sigma$  x = ? $\sigma$  y
  from x y have x: (x, ? $\sigma$  x)  $\in$  set ?d and y: (y, ? $\sigma$  y)  $\in$  set ?d
  unfolding mk-subst-domain by auto
  from x obtain i where di: ?d ! i = (x, ? $\sigma$  x) and i: i < length ?d unfolding
set-conv-nth by auto
  from y obtain j where dj: ?d ! j = (y, ? $\sigma$  y) and j: j < length ?d unfolding
set-conv-nth by auto
  from di i have mi: ?m ! i = ? $\sigma$  x by simp
  from dj j have mj: ?m ! j = ? $\sigma$  x unfolding id by simp
  from mi mj have id: ?m ! i = ?m ! j by simp
  from dist[unfolded distinct-conv-nth] i j id have id: i = j by auto
  with di dj
  show x = y by auto
qed
qed
finally
show ?thesis unfolding is-renaming-def by simp
qed

```

definition *is-inverse-renaming-impl* :: (*f*, *v*) *substL* \Rightarrow (*f*, *v*) *substL* **where**
is-inverse-renaming-impl $\sigma \equiv$
let $\sigma' = \text{mk-subst-domain } \sigma$ *in*
map ($\lambda (x, y). (\text{the-Var } y, \text{Var } x)$) σ'

lemma *is-inverse-renaming-impl* [*simp*]:
fixes $\sigma :: (\text{'f}, \text{'v}) \text{ substL}$
assumes *var*: *is-renaming* (*mk-subst* *Var* σ)
shows *mk-subst* *Var* (*is-inverse-renaming-impl* σ) = *is-inverse-renaming* (*mk-subst* *Var* σ) (**is** ?*l* = ?*r*)
proof (*rule ext*)
fix *x*
let ? σ = *mk-subst* *Var* σ
let ? σ' = *mk-subst-domain* σ
let ?*m* = *map* ($\lambda (x, y). (\text{the-Var } y, \text{Var } x :: (\text{'f}, \text{'v}) \text{ term}))$) ? σ'
let ?*ran* = *subst-range* ? σ
note *d* = *is-inverse-renaming-impl-def is-inverse-renaming-def*
{
fix *t*
assume (*x*, *t*) \in *set* ?*m*
then obtain *u z* **where** *id*: (*x*, *t*) = (*the-Var* *u*, *Var* *z*) **and** *mem*: (*z*, *u*) \in *set* ? σ' **by** *auto*
from *var*[*unfolded is-renaming-def*] *mem* **obtain** *zz* **where** *u*: *u* = *Var* *zz*
unfolding *mk-subst-domain* **by** *auto*
from *id*[*unfolded u*] **have** *id*: *zz* = *x* *t* = *Var* *z* **by** *auto*

with *mem* *u* **have** $(z, \text{Var } x) \in \text{set } ?\sigma'$ **by** *auto*
then **have** $? \sigma \ z = \text{Var } x \ z \in \text{subst-domain } ? \sigma$ **unfolding** *mk-subst-domain* **by**
auto
with *id* **have** $\exists z. t = \text{Var } z \wedge ? \sigma \ z = \text{Var } x \wedge z \in \text{subst-domain } ? \sigma$ **by** *auto*
} **note** *one = this*
have $?l \ x = \text{mk-subst } \text{Var } ?m \ x$ **unfolding** *d* **by** *simp*
also **have** $\dots = ?r \ x$
proof (*cases* $\text{Var } x \in ?ran$)
 case *False*
 {
 fix *t*
 assume $(x, t) \in \text{set } ?m$
 from *one*[*OF this*] **obtain** *z* **where** *t*: $t = \text{Var } z$ **and** *z*: $? \sigma \ z = \text{Var } x$
 and *dom*: $z \in \text{subst-domain } ? \sigma$ **by** *auto*
 from *z dom False* **have** *False* **by** *force*
 }
from *this*[*OF map-of-SomeD*[*of ?m x*]] **have** *look*: $\text{map-of } ?m \ x = \text{None}$
 by (*cases map-of ?m x, auto*)
then **have** $\text{mk-subst } \text{Var } ?m \ x = \text{Var } x$ **unfolding** *mk-subst-def* **by** *auto*
also **have** $\dots = ?r \ x$ **using** *False* **unfolding** *d* **by** *simp*
finally **show** *?thesis* .
next
 case *True*
 then **obtain** *y* **where** *y*: $y \in \text{subst-domain } ? \sigma$ **and** *x*: $? \sigma \ y = \text{Var } x$ **by** *auto*
 then **have** $(y, \text{Var } x) \in \text{set } ? \sigma'$ **unfolding** *mk-subst-domain* **by** *auto*
 then **have** $(x, \text{Var } y) \in \text{set } ?m$ **by** *force*
 then **obtain** *u* **where** *look*: $\text{map-of } ?m \ x = \text{Some } u$ **using** *map-of-eq-None-iff*[*of ?m x*]
 by (*cases map-of ?m x, force+*)
 from *map-of-SomeD*[*OF this*] **have** *xu*: $(x, u) \in \text{set } ?m$ **by** *auto*
 from *one*[*OF this*] **obtain** *z* **where** *u*: $u = \text{Var } z$ **and** *z*: $? \sigma \ z = \text{Var } x$ **and**
 dom: $z \in \text{subst-domain } ? \sigma$ **by** *auto*
 have $\text{mk-subst } \text{Var } ?m \ x = \text{Var } z$ **unfolding** *mk-subst-def* *look u* **by** *simp*
 also **have** $\dots = ?r \ x$ **using** *is-renaming-inverse-domain*[*OF var dom*] *z* **by** *auto*
 finally **show** *?thesis* .
qed
finally **show** $?l \ x = ?r \ x$.
qed

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* σ τ = *subst-compose-impl* (*map* $(\lambda x. (x, \sigma \ x))$ *xs*) τ

lemma *mk-subst-case* [*simp*]:

mk-subst *Var* (*mk-subst-case* *xs* σ τ) =

$(\lambda x. \text{if } x \in \text{set } xs \text{ then } \sigma \ x \cdot \text{mk-subst } \text{Var } \tau \text{ else } \text{mk-subst } \text{Var } \tau \ x)$

proof –

let $?m = \text{map } (\lambda x. (x, \sigma \ x)) \ xs$

```

have id: mk-subst Var ?m = ( $\lambda x.$  if  $x \in \text{set } xs$  then  $\sigma x$  else  $\text{Var } x$ ) (is ?l = ?r)
proof (rule ext)
  fix x
  show ?l x = ?r x
  proof (cases x  $\in$  set xs)
    case True
      then have (x,  $\sigma x$ )  $\in$  set ?m by auto
      with map-of-eq-None-iff[of ?m x] obtain u where look: map-of ?m x = Some
u by auto
      from map-of-SomeD[OF look] have u: u =  $\sigma x$  by auto
      show ?thesis unfolding mk-subst-def look u using True by auto
    next
      case False
      with map-of-SomeD[of ?m x]
        have look: map-of ?m x = None by (cases map-of ?m x, auto)
        show ?thesis unfolding mk-subst-def look using False by auto
      qed
    qed
  show ?thesis unfolding mk-subst-case-def subst-compose-impl id
  unfolding subst-compose-def by auto
qed

```

definition check-linear-term :: ('f :: showl, 'v :: showl) term \Rightarrow showsl check
where
 check-linear-term s = check (linear-term s)
 (showsl (STR "the term ") \circ showsl s \circ showsl (STR " is not linear" $\boxed{\longleftrightarrow}$ '))

lemma check-linear-term [simp]:
 isOK (check-linear-term s) = linear-term s
by (simp add: check-linear-term-def)

definition check-ground-term :: ('f :: showl, 'v :: showl) term \Rightarrow showsl check
where
 check-ground-term s = check (ground s)
 (showsl (STR "the term ") \circ showsl s \circ showsl (STR " is not a ground
 term" $\boxed{\longleftrightarrow}$ '))

lemma check-ground-term [simp]:
 isOK (check-ground-term s) \longleftrightarrow ground s
by (simp add: check-ground-term-def)

type-synonym 'f sig-list = ('f \times nat)list

fun check-funas-term :: 'f :: showl sig \Rightarrow ('f, 'v :: showl)term \Rightarrow showsl check **where**
 check-funas-term F (Fun f ts) = do {
 check ((f, length ts) \in F) (showsl (Fun f ts)
 o showsl-lit (STR "problem: root of subterm ") o showsl f o showsl-lit (STR
 " not in signature" $\boxed{\longleftrightarrow}$ '));

```

      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  $\subseteq$ 
F)
  by (induct t, auto)

end

```

13 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)
  by (simp add: subst-list-def subst-set-def)

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  $\cdot$   $\delta$ )

proof -
  let ? $\sigma$  =  $\lambda$  x. if x  $\in$  S then  $\sigma$  x else  $\tau$  ((the-inv T) x)
  let ?t = map-vars-term T t
  have ids: s  $\cdot$   $\sigma$  = s  $\cdot$  ? $\sigma$ 
    by (rule term-subst-eq, insert s, auto)
  have t  $\cdot$   $\tau$  = map-vars-term (the-inv T) ?t  $\cdot$   $\tau$ 
    unfolding map-vars-term-compose o-def using the-inv-f-f[OF inj] by (auto
simp: term.map-ident)
  also have ... = ?t  $\cdot$  ( $\tau$   $\circ$  the-inv T) unfolding apply-subst-map-vars-term ..
  also have ... = ?t  $\cdot$  ? $\sigma$ 
  proof (rule term-subst-eq)
    fix x
    assume x  $\in$  vars-term ?t
    then have x  $\in$  T ' UNIV unfolding term.set-map by auto
    then have x  $\notin$  S using ST by auto
    then show ( $\tau$   $\circ$  the-inv T) x = ? $\sigma$  x by simp
  qed

```

finally have $idt: t \cdot \tau = ?t \cdot ?\sigma$ **by** *simp*
from $id[unfolds\ id\ idt]$ **have** $id: s \cdot ?\sigma = ?t \cdot ?\sigma$.
with $mgu-complete[of\ s\ ?t]$ id **obtain** μ **where** $\mu: mgu\ s\ ?t = Some\ \mu$
unfolding *unifiers-def* **by** (*cases mgu s ?t, auto*)
from $the-mgu[OF\ id]$ **have** $id: s \cdot \mu = map-vars-term\ T\ t \cdot \mu$ **and** $\sigma: ?\sigma = \mu \circ_s$
 $?\sigma$
unfolding *the-mgu-def* μ **by** *auto*
have $s \cdot \sigma = s \cdot (\mu \circ_s ?\sigma)$ **unfolding** *ids* **using** σ **by** *simp*
also have $\dots = s \cdot \mu \cdot ?\sigma$ **by** *simp*
finally have $ids: s \cdot \sigma = s \cdot \mu \cdot ?\sigma$.
{
 fix x
 have $\tau\ x = ?\sigma\ (T\ x)$ **using** *ST* **unfolding** *the-inv-f-f[OF inj]* **by** *auto*
 also have $\dots = (\mu \circ_s ?\sigma)\ (T\ x)$ **using** σ **by** *simp*
 also have $\dots = \mu\ (T\ x) \cdot ?\sigma$ **unfolding** *subst-compose-def* **by** *simp*
 finally have $\tau\ x = \mu\ (T\ x) \cdot ?\sigma$.
}
note $\tau = this$
{
 fix $t :: ('f, 'v)term$
 have $t \cdot \tau = t \cdot (\lambda\ x. \mu\ (T\ x) \cdot ?\sigma)$ **unfolding** $\tau[symmetric]$..
 also have $\dots = map-vars-term\ T\ t \cdot \mu \cdot ?\sigma$ **unfolding** *apply-subst-map-vars-term*

 subst-subst **by** (*rule term-subst-eq, simp add: subst-compose-def*)
 finally have $t \cdot \tau = map-vars-term\ T\ t \cdot \mu \cdot ?\sigma$.
}
note $idt = this$
{
 fix x
 assume $x \in S$
 then have $\sigma\ x = ?\sigma\ x$ **by** *simp*
 also have $\dots = (\mu \circ_s ?\sigma)\ x$ **using** σ **by** *simp*
 also have $\dots = \mu\ x \cdot ?\sigma$ **unfolding** *subst-compose-def* ..
 finally have $\sigma\ x = \mu\ x \cdot ?\sigma$.
}
note $\sigma = this$
show *?thesis*
 by (*rule exI[of - μ], rule exI[of - $?\sigma$], insert $\mu\ ids\ idt\ \sigma$, auto)*

qed

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
 $(\forall\ x \in range\ x-var \cup range\ z-var. \sigma\ x = \mu\ x \cdot \delta)$

```

proof –
  have inj: inj y-var unfolding inj-on-def by simp
  show ?thesis
    by (rule mgu-var-disjoint-right[OF s inj - id], auto)
qed

lemma not-elem-subst-of:
  assumes  $x \notin \text{set } (\text{map } \text{fst } xs)$ 
  shows  $(\text{subst-of } xs) \ x = \text{Var } x$ 
  using assms proof (induct xs)
  case (Cons y xs)
  then show ?case unfolding subst-of-simps
    by (metis Term.term.simps(17) insert-iff list.simps(15) list.simps(9) singletonD
subst-compose subst-ident)
qed simp

lemma subst-of-id:
  assumes  $\bigwedge s. s \in (\text{set } ss) \longrightarrow (\exists x \ t. s = (x, t) \wedge t = \text{Var } x)$ 
  shows  $\text{subst-of } ss = \text{Var}$ 
using assms proof (induct ss)
  case (Cons s ss)
  then obtain y t where  $s:s = (y, t)$  and  $t:t = \text{Var } y$ 
    by (metis list.set-intros(1))
  from Cons have  $\text{subst-of } ss = \text{Var}$ 
    by simp
  then show ?case
    unfolding subst-of-def foldr.simps o-apply s t by simp
qed simp

lemma subst-of-apply:
  assumes  $(x, t) \in \text{set } ss$ 
  and  $\forall (y, s) \in \text{set } ss. (y = x \longrightarrow s = t)$ 
  and  $\text{set } (\text{map } \text{fst } ss) \cap \text{vars-term } t = \{\}$ 
  shows  $\text{subst-of } ss \ x = t$ 
  using assms proof (induct ss)
  case (Cons a ss)
  show ?case proof (cases  $(x, t) \in \text{set } ss$ )
    case True
    from Cons(1)[OF True] Cons(3,4) have sub:  $\text{subst-of } ss \ x = t$ 
      by (simp add: disjoint-iff)
    from Cons(2,4) have  $\text{fst } a \notin \text{vars-term } t$ 
      by fastforce
    then show ?thesis
      unfolding subst-of-simps subst-compose sub by simp
  next
  case False
  then have  $x \notin \text{set } (\text{map } \text{fst } ss)$ 
    using Cons(3) by auto

```



```

next
  case False
  then show ?thesis proof(cases  $x \notin \text{vars-term } t$ )
    case True
    let ? $\sigma$ =(subst  $x \ t$ )
    have subst:subst-list ? $\sigma$   $E = (\text{subst-list } ?\sigma \text{ es1}) @ (\text{fst } e \cdot ?\sigma, \text{snd } e \cdot ?\sigma)$ 
# (subst-list ? $\sigma$   $E2$ )
    unfolding  $E$  by (simp add: subst-list-def)
    from 3(2)[OF False True substs] 3(4) show ?thesis
    unfolding Cons  $e1$  append-Cons unify.simps using False True
    by (smt (verit, ccfv-SIG)  $E$  fst-eqD snd-eqD subst-list-append substs)
  next
  case False
  then show ?thesis
    unfolding Cons  $e1$  append-Cons unify.simps using 3 Cons by auto
  qed
qed
qed
next
  case ( $4 \ f \ ts \ x \ E \ bs$ )
  show ?case proof(cases  $E1$ )
    case Nil
    with 4(3) show ?thesis
    by (simp add: unify-Cons-same)
  next
  case (Cons  $e1 \text{ es1}$ )
  with 4(2) have  $e1:e1 = (\text{Fun } f \ ts, \text{Var } x)$ 
  by simp
  with 4(2) Cons have  $E:E = \text{es1} @ e \# E2$ 
  by simp
  show ?thesis proof(cases  $x \notin \text{vars-term } (\text{Fun } f \ ts)$ )
    case True
    let ? $\sigma$ =(subst  $x \ (\text{Fun } f \ ts)$ )
    have subst:subst-list ? $\sigma$   $E = (\text{subst-list } ?\sigma \text{ es1}) @ (\text{fst } e \cdot ?\sigma, \text{snd } e \cdot ?\sigma) \#$ 
(subst-list ? $\sigma$   $E2$ )
    unfolding  $E$  by (simp add: subst-list-def)
    from 4(1)[OF True substs] 4(3) show ?thesis
    unfolding Cons  $e1$  append-Cons unify.simps using True
    by (metis  $E$  fst-conv snd-conv subst-list-append substs)
  next
  case False
  then show ?thesis
    unfolding Cons  $e1$  append-Cons unify.simps using 4 Cons by auto
  qed
qed
qed simp

```

lemma *unify-filter-same*:
shows *unify* (*filter* ($\lambda e. \text{fst } e \neq \text{snd } e$) E) $ys = \text{unify } E \ ys$

```

proof(induction length E arbitrary:E rule:full-nat-induct)
  case 1
  show ?case proof(cases E)
    case (Cons e es)
    then show ?thesis proof(cases filter (λe. fst e ≠ snd e) E = E)
      case False
      then obtain E1 e E2 where E:E = E1 @ e # E2 and eq:fst e = snd e
        by (meson filter-True split-list)
      with unify-equation-same have unify E ys = unify (E1 @ E2) ys
        by blast
      moreover from 1 E have unify (filter (λe. fst e ≠ snd e) (E1 @ E2)) ys =
unify (E1 @ E2) ys
        by (metis (no-types, lifting) add-Suc-right length-append length-nth-simps(2)
order-refl)
      moreover have filter (λe. fst e ≠ snd e) E = filter (λe. fst e ≠ snd e) (E1
@ E2)
        unfolding E using eq by auto
      ultimately show ?thesis
        by presburger
    qed simp
  qed simp
qed

```

```

lemma unify-ctxt-same:
  shows unify ((C⟨s⟩, C⟨t⟩)#xs) ys = unify ((s, t)#xs) ys
proof(induct C)
  case (More f ss1 C ss2)
  let ?us=zip (ss1 @ C⟨s⟩ # ss2) (ss1 @ C⟨t⟩ # ss2)
  have decomp:decompose (Fun f (ss1 @ C⟨s⟩ # ss2)) (Fun f (ss1 @ C⟨t⟩ # ss2))
  = Some ?us
    unfolding decompose-def by (simp add: zip-option-zip-conv)
  have unif:unify (((More f ss1 C ss2)⟨s⟩, (More f ss1 C ss2)⟨t⟩) # xs) ys = unify
  (?us @ xs) ys
    unfolding ctxt-apply-term.simps unify.simps decomp by simp
  have *:?us = (zip ss1 ss1) @ (C⟨s⟩, C⟨t⟩) # (zip ss2 ss2)
    by simp
  have filter-us:filter (λe. fst e ≠ snd e) ?us = filter (λe. fst e ≠ snd e) [(C⟨s⟩,
C⟨t⟩)]
    unfolding * filter-append filter.simps by (smt (verit, ccfv-SIG) filter-False
in-set-zip self-append-conv2)
  have filter (λe. fst e ≠ snd e) (?us@xs) = filter (λe. fst e ≠ snd e) ((C⟨s⟩,
C⟨t⟩)#xs)
    unfolding filter-append filter-us filter.simps by simp
  with More have unify (?us @ xs) ys = unify ((s, t)#xs) ys
    using unify-filter-same by (smt (verit, ccfv-threshold))
  with unif show ?case by simp
qed simp

```

14 Unification of linear and variable disjoint terms

definition $\text{left-substs} :: ('f, 'v) \text{ term} \Rightarrow ('f, 'w) \text{ term} \Rightarrow ('v \times ('f, 'w) \text{ term}) \text{ list}$
where $\text{left-substs } s \ t = (\text{let } \text{filtered-vars} = \text{filter } (\lambda(-, p). p \in \text{poss } t) (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } s)))$
 $\text{in } \text{map } (\lambda(x, p). (x, t|-p)) \text{ filtered-vars}$

definition $\text{right-substs} :: ('f, 'v) \text{ term} \Rightarrow ('f, 'w) \text{ term} \Rightarrow ('w \times ('f, 'v) \text{ term}) \text{ list}$
where $\text{right-substs } s \ t = (\text{let } \text{filtered-vars} = \text{filter } (\lambda(-, q). q \in \text{fun-poss } s) (\text{zip } (\text{vars-term-list } t) (\text{var-poss-list } t)))$
 $\text{in } \text{map } (\lambda(y, q). (y, s|-q)) \text{ filtered-vars}$

abbreviation $\text{linear-unifier } s \ t \equiv \text{subst-of } ((\text{left-substs } s \ t) @ (\text{right-substs } s \ t))$

lemma $\text{left-substs-imp-props}$:

assumes $(x, u) \in \text{set } (\text{left-substs } s \ t)$

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 –

from assms **obtain** p **where** $1:(x, p) \in \text{set } (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } s))$ **and** $2:p \in \text{poss } t \wedge t|-p = u$

unfolding left-substs-def Let-def **using** Pair-inject case-prodE filter-set in-set-id length-map map-nth-eq-conv member-filter nth-mem old.prod.case **by** auto

from 1 **have** $p:p \in \text{poss } s$

by $(\text{metis } \text{set-} \text{zip-rightD } \text{var-poss-imp-poss } \text{var-poss-list-sound})$

from 1 **obtain** i **where** $i < \text{length } (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } s))$ **and** $(\text{vars-term-list } s)![i] = x$ **and** $(\text{var-poss-list } s)![i] = p$

by $(\text{smt } (z3) \text{ Pair-inject length-} \text{zip mem-Collect-eq set-} \text{zip})$

then have $s|-p = \text{Var } x$

by $(\text{metis } \text{length-} \text{zip min-less-iff-conj } \text{vars-term-list-var-poss-list})$

with $2 \ p$ **show** $?thesis$

by blast

qed

lemma $\text{props-imp-left-substs}$:

assumes $p \in \text{poss } s$ **and** $s|-p = \text{Var } x$ **and** $p \in \text{poss } t$ **and** $t|-p = u$

shows $(x, u) \in \text{set } (\text{left-substs } s \ t)$

proof –

from assms **obtain** i **where** $(\text{var-poss-list } s)![i] = p$ **and** $(\text{vars-term-list } s)![i] = x$

by $(\text{metis } \text{in-set-conv-nth length-var-poss-list term.inject(1) var-poss-iff var-poss-list-sound vars-term-list-var-poss-list})$

then have $(x, p) \in \text{set } (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } s))$

by $(\text{metis } \text{assms(1) assms(2) in-set-id in-set-} \text{zip length-var-poss-list prod.sel(1) prod.sel(2) term.inject(1) var-poss-iff var-poss-list-sound vars-term-list-var-poss-list})$

with assms(3) **have** $(x, p) \in \text{set } (\text{filter } (\lambda(-, p). p \in \text{poss } t) (\text{zip } (\text{vars-term-list } s) (\text{var-poss-list } s)))$

by simp

then show $?thesis$ **unfolding** left-substs-def Let-def assms(4) $[\text{symmetric}]$

by $(\text{smt } (z3) \text{ case-prod-conv in-set-conv-nth length-map map-nth-eq-conv})$

qed

lemma *right-substs-imp-props*:

assumes $(x, u) \in \text{set } (\text{right-substs } s \ t)$

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–

from *assms* **obtain** q **where** $1:(x, q) \in \text{set } (\text{zip } (\text{vars-term-list } t)(\text{var-poss-list } t))$ **and** $2:q \in \text{fun-poss } s|-q = u$

unfolding *right-substs-def Let-def* **using** *Pair-inject case-prodE filter-set in-set-idx length-map map-nth-eq-conv member-filter nth-mem old.prod.case* **by** *auto*

from 1 **have** $q:q \in \text{poss } t$

by (*metis set-zip-rightD var-poss-imp-poss var-poss-list-sound*)

from 1 **obtain** i **where** $i < \text{length } (\text{zip } (\text{vars-term-list } t)(\text{var-poss-list } t))$ **and** $(\text{vars-term-list } t)!i = x$ **and** $(\text{var-poss-list } t)!i = q$

by (*smt (z3) Pair-inject length-zip mem-Collect-eq set-zip*)

then have $t|-q = \text{Var } x$

by (*metis length-zip min-less-iff-conj vars-term-list-var-poss-list*)

with 2 q **show** *?thesis*

by *blast*

qed

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 $(x, u) \in \text{set } (\text{right-substs } s \ t)$

proof–

from *assms* **obtain** i **where** $(\text{var-poss-list } t)!i = q$ **and** $(\text{vars-term-list } t)!i = x$

by (*metis in-set-conv-nth length-var-poss-list term.inject(1) var-poss-iff var-poss-list-sound vars-term-list-var-poss-list*)

then have $(x, q) \in \text{set } (\text{zip } (\text{vars-term-list } t)(\text{var-poss-list } t))$

by (*metis assms(3) assms(4) in-set-conv-nth in-set-zip length-var-poss-list prod.sel(1) prod.sel(2) term.inject(1) var-poss-iff var-poss-list-sound vars-term-list-var-poss-list*)

with *assms(1)* **have** $(x, q) \in \text{set } (\text{filter } (\lambda(-, p). p \in \text{fun-poss } s) (\text{zip } (\text{vars-term-list } t) (\text{var-poss-list } t)))$

by *simp*

then show *?thesis* **unfolding** *right-substs-def Let-def* *assms(2)[symmetric]*

by (*smt (z3) case-prod-conv in-set-conv-nth length-map map-nth-eq-conv*)

qed

lemma *map-fst-left-substs*:

$\text{set } (\text{map fst } (\text{left-substs } s \ t)) \subseteq \text{vars-term } s$

unfolding *left-substs-def* **using** *zip-fst* **by** *fastforce*

lemma *map-snd-left-substs*:

assumes $t' \in \text{set } (\text{map snd } (\text{left-substs } s \ t))$

shows $\text{vars-term } t' \subseteq \text{vars-term } t$

proof–

from *assms* **obtain** x **where** $(x, t') \in \text{set } (\text{left-substs } s \ t)$

by *force*

then show *?thesis*

using *left-substs-imp-props* by (*metis vars-term-subt-at*)
qed

lemma *map-fst-right-substs*:
 $set (map fst (right-substs s t)) \subseteq vars-term t$
unfolding *right-substs-def* **using** *zip-fst* **by** *fastforce*

lemma *map-snd-right-substs*:
assumes $t' \in set (map snd (right-substs s t))$
shows $vars-term t' \subseteq vars-term s$
proof–
from *assms* **obtain** x **where** $(x, t') \in set (right-substs s t)$
by *force*
then show *?thesis*
using *right-substs-imp-props* **by** (*metis fun-poss-imp-poss vars-term-subt-at*)
qed

lemma *distinct-map-fst-left-substs*:
assumes *linear-term t*
shows *distinct (map fst (left-substs t s))*
proof–
from *linear-term-distinct-vars[OF assms]* **have** *dist:distinct (map fst (filter ($\lambda(x, p). p \in poss s$) (zip (vars-term-list t) (var-poss-list t))))*
by (*simp add: distinct-map-filter length-var-poss-list*)
have $map fst (left-substs t s) = (map fst (filter (\lambda(x, p). p \in poss s) (zip (vars-term-list t) (var-poss-list t))))$
unfolding *left-substs-def Let-def* **by** *auto*
with *dist* **show** *?thesis*
by *presburger*
qed

lemma *distinct-map-fst-right-substs*:
assumes *linear-term t*
shows *distinct (map fst (right-substs s t))*
proof–
from *linear-term-distinct-vars[OF assms]* **have** *dist:distinct (map fst (filter ($\lambda(x, p). p \in fun-poss s$) (zip (vars-term-list t) (var-poss-list t))))*
by (*simp add: distinct-map-filter length-var-poss-list*)
have $map fst (right-substs s t) = (map fst (filter (\lambda(x, p). p \in fun-poss s) (zip (vars-term-list t) (var-poss-list t))))$
unfolding *right-substs-def Let-def* **by** *auto*
with *dist* **show** *?thesis*
by *presburger*
qed

lemma *is-partition-map-snd-left-substs*:
assumes *linear-term s linear-term t*
shows *is-partition (map (vars-term \circ snd) (left-substs t s))*
proof–

```

{fix i j assume j:j < length (left-substs t s) and i:i < j
  from i j obtain x u where xu:(x, u) = (left-substs t s)!i
    by (metis surj-pair)
  from i j obtain y v where yv:(y, v) = (left-substs t s)!j
    by (metis surj-pair)
  from xu i j obtain p where p:p ∈ poss t t|-p = Var x p ∈ poss s s|-p = u
    using left-substs-imp-props by (metis Suc-lessD less-trans-Suc nth-mem)
  from yv i j obtain q where q:q ∈ poss t t|-q = Var y q ∈ poss s s|-q = v
    using left-substs-imp-props by (metis nth-mem)
  from assms(2) have distinct (map fst (left-substs t s))
    using distinct-map-fst-left-substs by blast
  with xu yv i j have x ≠ y
    by (metis (mono-tags, lifting) Suc-lessD distinct-map eq-key-imp-eq-value
less-trans-Suc nat-neq-iff nth-eq-iff-index-eq nth-mem)
  with p(1,2) q(1,2) have p ⊥ q
    by (metis term.inject(1) var-poss-iff var-poss-parallel)
  with assms(1) p(3,4) q(3,4) have vars-term (snd ((left-substs t s)!i)) ∩
vars-term (snd ((left-substs t s)!j)) = {}
    by (metis linear-subterms-disjoint-vars snd-eqD xu yv)
}
then show ?thesis unfolding is-partition-def map-map[symmetric] by auto
qed

```

lemma is-partition-map-snd-right-substs:

```

assumes linear-term s linear-term t
shows is-partition (map (vars-term ∘ snd) (right-substs t s))
proof-
{fix i j assume j:j < length (right-substs t s) and i:i < j
  from i j obtain x u where xu:(x, u) = (right-substs t s)!i
    by (metis surj-pair)
  from i j obtain y v where yv:(y, v) = (right-substs t s)!j
    by (metis surj-pair)
  from xu i j obtain p where p:p ∈ poss s s|-p = Var x p ∈ fun-poss t t|-p
= u
    using right-substs-imp-props by (metis Suc-lessD less-trans-Suc nth-mem)
  from yv i j obtain q where q:q ∈ poss s s|-q = Var y q ∈ fun-poss t t|-q =
v
    using right-substs-imp-props by (metis nth-mem)
  from assms(1) have distinct (map fst (right-substs t s))
    using distinct-map-fst-right-substs by blast
  with xu yv i j have x ≠ y
    by (metis (mono-tags, lifting) Suc-lessD distinct-map eq-key-imp-eq-value
less-trans-Suc nat-neq-iff nth-eq-iff-index-eq nth-mem)
  with p(1,2) q(1,2) have p ⊥ q
    by (metis term.inject(1) var-poss-iff var-poss-parallel)
  with assms(2) p(3,4) q(3,4) have vars-term (snd ((right-substs t s)!i)) ∩
vars-term (snd ((right-substs t s)!j)) = {}
    by (metis fun-poss-imp-poss linear-subterms-disjoint-vars snd-eqD xu yv)
}

```

then show *?thesis unfolding is-partition-def map-map[symmetric]* by auto
qed

lemma *distinct-fst-lsubsts-snd-rsubsts*:

assumes *linear-term s*

shows $(\text{set } (\text{map fst } (\text{left-substs } s \ t))) \cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{right-substs } s \ t))) = \{\}$

proof –

{fix *x u* assume $(x,u) \in \text{set } (\text{left-substs } s \ t)$
 then obtain *p* where $p:p \in \text{poss } s \mid -p = \text{Var } x \ p \in \text{poss } t \mid -p = u$
 by (meson *left-substs-imp-props*)
 {fix *y v* assume $(y,v) \in \text{set } (\text{right-substs } s \ t)$
 then obtain *q* where $q:q \in \text{poss } t \mid -q = \text{Var } y \ q \in \text{fun-poss } s \mid -q = v$
 by (meson *right-substs-imp-props*)
 with *p* have $p \perp q$
 by (metis *Term.term.simps(4)* *append.right-neutral fun-poss-fun-conv fun-poss-imp-poss parallel-pos prefix-pos-diff var-pos-maximal*)
 with *assms p(1,2) q(3,4)* have $x \notin \text{vars-term } v$
 using *fun-poss-imp-poss linear-subterms-disjoint-vars* by fastforce
 }
 then have $x \notin \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{right-substs } s \ t)))$
 by fastforce
 }
 then show *?thesis* by fastforce
 qed

lemma *distinct-fst-rsubsts-snd-lsubsts*:

assumes *linear-term t*

shows $(\text{set } (\text{map fst } (\text{right-substs } s \ t))) \cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{left-substs } s \ t))) = \{\}$

proof –

{fix *x u* assume $(x,u) \in \text{set } (\text{right-substs } s \ t)$
 then obtain *p* where $p:p \in \text{poss } t \mid -p = \text{Var } x \ p \in \text{fun-poss } s \mid -p = u$
 by (meson *right-substs-imp-props*)
 {fix *y v* assume $(y,v) \in \text{set } (\text{left-substs } s \ t)$
 then obtain *q* where $q:q \in \text{poss } s \mid -q = \text{Var } y \ q \in \text{poss } t \mid -q = v$
 by (meson *left-substs-imp-props*)
 with *p* have $p \perp q$
 by (metis *Term.term.simps(4)* *append.right-neutral fun-poss-fun-conv fun-poss-imp-poss parallel-pos prefix-pos-diff var-pos-maximal*)
 with *assms p(1,2) q(3,4)* have $x \notin \text{vars-term } v$
 using *fun-poss-imp-poss linear-subterms-disjoint-vars* by fastforce
 }
 then have $x \notin \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) (\text{left-substs } s \ t)))$
 by fastforce
 }
 then show *?thesis* by fastforce
 qed

```

lemma linear-unifier-same:
  shows (linear-unifier t t) = Var
proof –
  let ?vars-left = filter ( $\lambda(-, p). p \in \text{poss } t$ ) (zip (vars-term-list t) (var-poss-list t))
  have left: ?vars-left = zip (vars-term-list t) (var-poss-list t)
  by (metis (no-types, lifting) filter-True split-beta var-poss-imp-poss var-poss-list-sound
zip-snd)
  let ?vars-right = filter ( $\lambda(-, q). q \in \text{fun-poss } t$ ) (zip (vars-term-list t) (var-poss-list
t))
  have right: ?vars-right = []
  by (metis (mono-tags, lifting) DiffE filter-False poss-simps(4) split-beta var-poss-list-sound
zip-snd)
  {fix i assume i: i < length (left-substs t t)
    let ?xi = vars-term-list t ! i
    from i have i < length (vars-term-list t)
    unfolding left-substs-def Let-def length-map left by simp
    then have left-substs t t ! i = (?xi, Var ?xi)
    unfolding left-substs-def left Let-def nth-map[OF i[unfolded left-substs-def
Let-def length-map left]]
    by (simp add: length-var-poss-list vars-term-list-var-poss-list)
  } note left-subst = this
  {fix x
    from left-subst have subst-of (left-substs t t) x = Var x
    using subst-of-id by (metis left-substs-imp-props prod.collapse)
  }
  then show ?thesis
  unfolding right-substs-def right left-substs-def left by auto
qed

lemma linear-unifier-var1:
  shows linear-unifier (Var x) t = subst x t
proof –
  have left-substs (Var x) t = [(x, t)]
  unfolding left-substs-def Let-def vars-term-list.simps var-poss-list.simps by
simp
  moreover have right-substs (Var x) t = []
  unfolding right-substs-def by simp
  ultimately show ?thesis
  by simp
qed

lemma linear-unifier-var2:
  shows linear-unifier (Fun f ts) (Var x) = subst x (Fun f ts)
proof –
  have left-substs (Fun f ts) (Var x) = []
  unfolding left-substs-def Let-def poss.simps
  by (metis (no-types, lifting) case-prodE filter-False map-is-Nil-conv set-zip-rightD
singletonD subt-at.simps(1) term.distinct(1) var-poss-iff var-poss-list-sound)
  moreover have right-substs (Fun f ts) (Var x) = [(x, Fun f ts)]

```

unfolding *right-substs-def* by (*simp add: vars-term-list.simps(1)*)
 ultimately show *?thesis*
 by *simp*
 qed

lemma *linear-unifier-id*:
 assumes $x \notin \text{vars-term } s$ and $x \notin \text{vars-term } t$
 shows $(\text{linear-unifier } s \ t) \ x = \text{Var } x$
 using *assms* by (*metis (no-types, lifting) Set.basic-monos(7) eval-term.simps(1)*)
map-fst-left-substs map-fst-right-substs not-elem-subst-of subst-compose subst-of-append

lemma *vars-subst-of*:
 $\text{vars-subst } (\text{subst-of } ts) \subseteq \text{set } (\text{map } \text{fst } ts) \cup \bigcup (\text{set } (\text{map } (\text{vars-term} \circ \text{snd}) \ ts))$
proof (*induct ts*)
 case *Nil*
 show *?case* unfolding *subst-of-simps list.map vars-subst-def* by *simp*
next
 case (*Cons t ts*)
 have $\text{vars-subst } (\text{subst } (\text{fst } t) (\text{snd } t)) \subseteq \{\text{fst } t\} \cup (\text{vars-term } (\text{snd } t))$
 unfolding *vars-subst-def* by *auto*
 with *Cons* show *?case* unfolding *subst-of-simps using vars-subst-compose*
 by (*smt (verit, del-insts) Un-iff UnionI Union-mono comp-apply empty-iff insert-iff list.set-intros(1) list.simps(9) set-subset-Cons subset-iff*)
 qed

lemma *vars-subst-linear-unifier*: $\text{vars-subst } (\text{linear-unifier } s \ t) \subseteq \text{vars-term } s \cup \text{vars-term } t$
proof–
 have $\text{vars-subst } (\text{linear-unifier } s \ t) \subseteq (\text{vars-subst } (\text{subst-of } (\text{left-substs } s \ t))) \cup (\text{vars-subst } (\text{subst-of } (\text{right-substs } s \ t)))$
 unfolding *subst-of-append using vars-subst-compose* by *force*
 moreover have $\text{vars-subst } (\text{subst-of } (\text{left-substs } s \ t)) \subseteq \text{vars-term } s \cup \text{vars-term } t$
proof–
 {fix *i* assume $i < \text{length } (\text{left-substs } s \ t)$
 then have $\text{map } (\text{vars-term} \circ \text{snd}) (\text{left-substs } s \ t) \ ! \ i \subseteq \text{vars-term } t$
 using *map-snd-left-substs nth-mem* by *fastforce*
 }
 then have $\bigcup (\text{set } (\text{map } (\text{vars-term} \circ \text{snd}) (\text{left-substs } s \ t))) \subseteq \text{vars-term } t$
 by (*metis Union-least in-set-conv-nth length-map*)
 then show *?thesis*
 using *vars-subst-of[of left-substs s t] map-fst-left-substs*
 by (*metis (no-types, lifting) subset-trans sup.mono*)
 qed
 moreover have $\text{vars-subst } (\text{subst-of } (\text{right-substs } s \ t)) \subseteq \text{vars-term } s \cup \text{vars-term } t$
proof–
 {fix *i* assume $i < \text{length } (\text{right-substs } s \ t)$

```

    then have map (vars-term  $\circ$  snd) (right-substs s t) ! i  $\subseteq$  vars-term s
    using map-snd-right-substs nth-mem by fastforce
  }
  then have  $\bigcup$  (set (map (vars-term  $\circ$  snd) (right-substs s t)))  $\subseteq$  vars-term s
  by (metis Union-least in-set-conv-nth length-map)
  then show ?thesis
  using vars-subst-of[of right-substs s t] map-fst-right-substs by fastforce
qed
ultimately show ?thesis by blast
qed

```

lemma *decompose-is-partition-vars-subst*:

```

  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 ds:decompose (Fun f ss) (Fun g ts) = Some ds
  shows is-partition (map vars-subst (map ( $\lambda(s,t).$  linear-unifier s t) ds))
proof -
  from assms have zip:ds = zip ss ts and l:length ss = length ts
  using decompose-Some by blast+
  {fix i j assume j:j < length ss and i:i < j
   from i j obtain si ti where s-t-i:(si, ti) = ds ! i ss ! i = si ts ! i = ti
   using l zip by force
   from j obtain sj tj where s-t-j:(sj, tj) = ds ! j ss ! j = sj ts ! j = tj
   using l zip by force
   have vars-term si  $\cap$  vars-term tj = {}
   using i j s-t-i s-t-j disj l by fastforce
   moreover have vars-term si  $\cap$  vars-term sj = {}
   using lin(1) s-t-i s-t-j i j var-in-linear-args by fastforce
   moreover have vars-term ti  $\cap$  vars-term tj = {}
   using lin(2) s-t-i s-t-j i j l var-in-linear-args by fastforce
   moreover have vars-term ti  $\cap$  vars-term sj = {}
   using i j s-t-i s-t-j disj l by fastforce
   ultimately have vars-subst (linear-unifier si ti)  $\cap$  vars-subst (linear-unifier sj
tj) = {}
   using vars-subst-linear-unifier by (smt (verit, ccfv-threshold) Un-iff disjoint-iff
in-mono)
   then have vars-subst (map ( $\lambda(s,t).$  linear-unifier s t) ds ! i)  $\cap$  vars-subst (map
( $\lambda(s,t).$  linear-unifier s t) ds ! j) = {}
   using i j s-t-j s-t-i l zip by auto
  }
  then show ?thesis unfolding is-partition-def map-map[symmetric] length-map
zip using l by auto
qed

```

lemma *compose-exists-subst*:

```

  assumes compose  $\sigma$  s x  $\neq$  Var x
  shows  $\exists i < \text{length } \sigma s. (\forall j < i. (\sigma s!j) x = \text{Var } x) \wedge (\sigma s!i) x \neq \text{Var } x$ 
  using assms proof(induct  $\sigma$  s)
  case (Cons  $\sigma$   $\sigma$  s)

```

```

then show ?case proof(cases  $\sigma x = \text{Var } x$ )
  case True
  from Cons(2) have  $\text{compose } \sigma s x \neq \text{Var } x$ 
    unfolding compose-simps subst-compose True by simp
  with Cons(1) obtain i where  $i < \text{length } \sigma s \ \forall j < i. (\sigma s ! j) x = \text{Var } x \ (\sigma s ! i)$ 
 $x \neq \text{Var } x$  by blast
  with True have  $\forall j < \text{Suc } i. ((\sigma \# \sigma s) ! j) x = \text{Var } x$ 
    by (metis less-Suc-eq-0-disj nth-Cons-0 nth-Cons-Suc)
  with i show ?thesis by auto
qed auto
qed simp

lemma subst-of-exists-binding:
  assumes subst-of xs y  $\neq$  Var y
  shows  $\exists i < \text{length } xs. \text{fst } (xs ! i) = y \wedge (\forall x \in \text{set } (\text{drop } (i+1) \ xs). \text{fst } x \neq y)$ 
  using assms proof(induct xs rule:rev-induct)
  case (snoc x xs)
  then show ?case proof(cases  $\text{fst } x = y$ )
    case False
    with snoc(2) have  $\text{subst-of } xs y \neq \text{Var } y$ 
      unfolding subst-of-append subst-compose
      by (metis (no-types, lifting) empty-iff eval-term.simps(1) insert-iff subst-compose
subst-ident subst-of-simps(1,3) term.set(3))
    with snoc(1) obtain i where  $i < \text{length } xs \ \text{fst } (xs ! i) = y \ \forall z \in \text{set } (\text{drop}$ 
 $(i+1) \ xs). \text{fst } z \neq y$  by blast
    from i(1) have  $\text{drop } (i+1) \ (xs @ [x]) = \text{drop } (i+1) \ xs @ [x]$  by auto
    with i(3) False have  $\forall z \in \text{set } (\text{drop } (i+1) \ (xs @ [x])). \text{fst } z \neq y$  by simp
    with i(1,2) show ?thesis
      by (metis append-Cons-nth-left length-append-singleton less-Suc-eq-le less-imp-le-nat)
    qed auto
  qed simp

lemma linear-unifier-obtain-binding:
  assumes disj:vars-term s  $\cap$  vars-term t = {} and lin-s:linear-term s and lin-t:linear-term
t
  and u:(linear-unifier s t) x = u  $u \neq \text{Var } x$ 
  shows  $(x \in \text{vars-term } s \wedge (x,u) \in \text{set } (\text{left-substs } s \ t)) \vee (x \in \text{vars-term } t \wedge$ 
 $(x,u) \in \text{set } (\text{right-substs } s \ t))$ 
proof–
  consider  $x \in \text{vars-term } s \mid x \in \text{vars-term } t \mid x \notin \text{vars-term } s \wedge x \notin \text{vars-term } t$ 
  by fastforce
  then show ?thesis proof(cases)
    case 1
    with disj have  $x \notin \text{vars-term } t$  by blast
    then have  $\text{right:subst-of } (\text{right-substs } s \ t) x = \text{Var } x$ 
      by (meson in-mono map-fst-right-substs not-elem-subst-of)
    with u have  $\text{subst-of } (\text{left-substs } s \ t) x \neq \text{Var } x$ 
      by (simp add: subst-compose)
    then obtain i u' where  $i < \text{length } (\text{left-substs } s \ t) \ (\text{left-substs } s \ t) ! i = (x,$ 

```

$u') \forall \text{subst} \in \text{set} (\text{drop } (i+1) (\text{left-substs } s \ t)). \text{fst } \text{subst} \neq x$
using *subst-of-exists-binding* **by** (*metis* (*mono-tags*, *opaque-lifting*) *eq-fst-iff*)
then obtain $l1 \ l2$ **where** $l1:l1 = \text{take } i (\text{left-substs } s \ t)$ **and** $l2:l2 = \text{drop } (i+1) (\text{left-substs } s \ t)$
and $l1l2:\text{left-substs } s \ t = l1 \ @ \ [(x,u')] \ @ \ l2$ **using** *id-take-nth-drop* **by** *fastforce*

from $i(\mathcal{J})$ **have** $l2\text{-subst}:\text{subst-of } l2 \ x = \text{Var } x$ **unfolding** $l2$ **by** (*meson* *nth-mem* *subst-of-exists-binding*)
then have $1:\text{subst-of } (\text{left-substs } s \ t) \ x = u' \cdot (\text{subst-of } l1)$
unfolding $l1l2$ *subst-of-append* *subst-compose* $l2\text{-subst}$ *eval-term.simps* **by** *simp*
from $i(1,2)$ **obtain** p **where** $p:p \in \text{poss } t \mid p = u'$ **using** *left-substs-imp-props*
by (*metis* *nth-mem*)
from *disj* p **have** $\text{set} (\text{map } \text{fst} (\text{left-substs } s \ t)) \cap (\text{vars-term } u') = \{\}$
by (*meson* *disjoint-iff* *map-fst-left-substs* *subsetD* *vars-term-subt-at*)
then have $2:u' \cdot (\text{subst-of } l1) = u'$
unfolding $l1$ **by** (*smt* (*verit*, *best*) *disjoint-iff* *in-set-takeD* *not-elem-subst-of* *subst-apply-term-empty* *take-map* *term-subst-eq*)
then have $u':\text{subst-of } (\text{left-substs } s \ t) \ x = u'$
using $1 \ 2$ **by** *simp*
from $i(1,2)$ **have** $u'\text{-elem}:(x, u') \in \text{set} (\text{left-substs } s \ t)$ **by** (*metis* *nth-mem*)
with $u' \ u$ **show** *?thesis*
unfolding *subst-of-append* *subst-compose* *right* *eval-term.simps*
by (*meson* *map-fst-left-substs* *not-elem-subst-of* *subset-iff*)
next
case 2
with *disj* **have** $x \notin \text{vars-term } s$ **by** *blast*
then have $\text{subst-of } (\text{left-substs } s \ t) \ x = \text{Var } x$
by (*meson* *in-mono* *map-fst-left-substs* *not-elem-subst-of*)
with u **have** $\text{subst-of } (\text{right-substs } s \ t) \ x \neq \text{Var } x$
by (*metis* *subst-compose* *subst-monoid-mult.mult* *left-neutral* *subst-of-append*)
then obtain $i \ u'$ **where** $i:i < \text{length} (\text{right-substs } s \ t)$ $(\text{right-substs } s \ t)!i = (x, u')$ $\forall \text{subst} \in \text{set} (\text{drop } (i+1) (\text{right-substs } s \ t)). \text{fst } \text{subst} \neq x$
using *subst-of-exists-binding* **by** (*metis* (*mono-tags*, *opaque-lifting*) *eq-fst-iff*)
then obtain $l1 \ l2$ **where** $l1:l1 = \text{take } i (\text{right-substs } s \ t)$ **and** $l2:l2 = \text{drop } (i+1) (\text{right-substs } s \ t)$
and $l1l2:\text{right-substs } s \ t = l1 \ @ \ [(x,u')] \ @ \ l2$ **using** *id-take-nth-drop* **by** *fastforce*
from $i(\mathcal{J})$ **have** $l2\text{-subst}:\text{subst-of } l2 \ x = \text{Var } x$ **unfolding** $l2$ **by** (*meson* *nth-mem* *subst-of-exists-binding*)
then have $1:\text{subst-of } (\text{right-substs } s \ t) \ x = u' \cdot (\text{subst-of } l1)$
unfolding $l1l2$ *subst-of-append* *subst-compose* $l2\text{-subst}$ *eval-term.simps* **by** *simp*
from $i(1,2)$ **obtain** p **where** $p:p \in \text{poss } s \mid p = u'$
using *right-substs-imp-props* **by** (*metis* *fun-poss-imp-poss* *nth-mem*)
from *disj* p **have** $\text{set} (\text{map } \text{fst} (\text{right-substs } s \ t)) \cap (\text{vars-term } u') = \{\}$
by (*meson* *disjoint-iff* *map-fst-right-substs* *subsetD* *vars-term-subt-at*)
then have $2:u' \cdot (\text{subst-of } l1) = u'$
unfolding $l1$ **by** (*smt* (*verit*, *best*) *disjoint-iff* *in-set-takeD* *not-elem-subst-of*

```

subst-apply-term-empty take-map term-subst-eq)
  then have u':subst-of (right-substs s t) x = u'
    using 1 2 by simp
  from i(1,2) have u'-elem:(x, u') ∈ set (right-substs s t) by (metis nth-mem)
  then have set (map fst (left-substs s t)) ∩ (vars-term u') = {}
    using distinct-fst-lsubsts-snd-rsubsts[OF lin-s] by (smt (verit, ccfv-SIG)
Union-iff comp-apply disjoint-iff in-set-conv-nth length-map nth-map snd-conv)
  then have u' · (subst-of (left-substs s t)) = u'
    by (metis disjoint-iff not-elem-subst-of subst-apply-term-empty term-subst-eq)

  with u u'-elem show ?thesis
    unfolding subst-of-append subst-compose u' by (metis map-fst-right-substs
not-elem-subst-of subset-eq u')
  next
  case 3
  then have x ∉ set (map fst ((left-substs s t) @ (right-substs s t)))
    using map-fst-left-substs map-fst-right-substs by fastforce
  then have (linear-unifier s t) x = Var x
    by (meson not-elem-subst-of)
  with u show ?thesis by simp
qed
qed

```

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)) = (⋃ e∈set ds. set (left-substs (fst
e) (snd e))) (is ?left = ?right)
proof
  from assms have ds:ds = zip ss ts
    using decompose-Some by auto
  show ?left ⊆ ?right proof
    fix x t assume (x,t) ∈ set (left-substs (Fun f ss) (Fun g ts))
    then obtain p where 1:p ∈ poss (Fun f ss) and 2:(Fun f ss)|-p = Var x and
3:p ∈ poss (Fun g ts) and 4:(Fun g ts)|-p = t
      by (meson left-substs-imp-props)
    from 1 2 obtain j p' where j1:j < length ss and p = j#p' and p' ∈ poss
(ss!j) and (ss!j)|-p' = Var x
      by auto
    moreover with 3 4 have j2:j < length ts and p' ∈ poss (ts!j) and (ts!j)|-p'
= t
      by auto
    ultimately have (x,t) ∈ set (left-substs (ss!j) (ts!j))
      by (meson props-imp-left-substs)
    moreover have ((ss!j),(ts!j)) ∈ set ds
      unfolding ds using j1 j2 by (metis length-zip min-less-iff-conj nth-mem
nth-zip)
    ultimately show (x,t) ∈ (⋃ e∈set ds. set (left-substs (fst e) (snd e)))

```

```

    by force
  qed
  show ?right  $\subseteq$  ?left proof
    fix  $x\ t$  assume  $(x,t) \in (\bigcup e \in \text{set } ds. \text{set } (\text{left-substs } (\text{fst } e) (\text{snd } e)))$ 
    then obtain  $j$  where  $j1:j < \text{length } ss$  and  $j2:j < \text{length } ts$  and  $(x,t) \in \text{set } (\text{left-substs } (ss!j) (ts!j))$ 
    unfolding  $ds$  by (metis (no-types, lifting) UN-E in-set-zip)
    then obtain  $p$  where  $1:p \in \text{poss } (ss!j)$  and  $2:(ss!j)|-p = \text{Var } x$  and  $3:p \in \text{poss } (ts!j)$  and  $4:(ts!j)|-p = t$ 
    by (meson left-substs-imp-props)
    then have  $j\#p \in \text{poss } (\text{Fun } f\ ss)$  and  $(\text{Fun } f\ ss)|-(j\#p) = \text{Var } x$  and  $(j\#p) \in \text{poss } (\text{Fun } g\ ts)$  and  $(\text{Fun } g\ ts)|-(j\#p) = t$ 
    using  $j1\ j2$  by auto
    then show  $(x,t) \in \text{set } (\text{left-substs } (\text{Fun } f\ ss) (\text{Fun } g\ ts))$ 
    by (meson props-imp-left-substs)
  qed
  qed

lemma decompose-right-substs:
  assumes  $\text{decompose } (\text{Fun } f\ ss) (\text{Fun } g\ ts) = \text{Some } ds$ 
  shows  $\text{set } (\text{right-substs } (\text{Fun } f\ ss) (\text{Fun } g\ ts)) = (\bigcup e \in \text{set } ds. \text{set } (\text{right-substs } (\text{fst } e) (\text{snd } e)))$  (is ?left = ?right)
proof
  from  $assms$  have  $ds:ds = \text{zip } ss\ ts$ 
  using  $\text{decompose-Some}$  by auto
  show ?left  $\subseteq$  ?right proof
    fix  $x\ t$  assume  $(x,t) \in \text{set } (\text{right-substs } (\text{Fun } f\ ss) (\text{Fun } g\ ts))$ 
    then obtain  $q$  where  $1:q \in \text{fun-poss } (\text{Fun } f\ ss)$  and  $2:(\text{Fun } f\ ss)|-q = t$  and  $3:q \in \text{poss } (\text{Fun } g\ ts)$  and  $4:(\text{Fun } g\ ts)|-q = \text{Var } x$ 
    by (meson right-substs-imp-props)
    from  $3\ 4$  obtain  $j\ q'$  where  $j1:j < \text{length } ts$  and  $q = j\#q'$  and  $q' \in \text{poss } (ts!j)$  and  $(ts!j)|-q' = \text{Var } x$ 
    by auto
    moreover with  $1\ 2$  have  $j2:j < \text{length } ss$  and  $q' \in \text{fun-poss } (ss!j)$  and  $(ss!j)|-q' = t$ 
    by auto
    ultimately have  $(x,t) \in \text{set } (\text{right-substs } (ss!j) (ts!j))$ 
    by (meson props-imp-right-substs)
    moreover have  $((ss!j), (ts!j)) \in \text{set } ds$ 
    unfolding  $ds$  using  $j1\ j2$  by (metis length-zip min-less-iff-conj nth-mem nth-zip)
    ultimately show  $(x,t) \in (\bigcup e \in \text{set } ds. \text{set } (\text{right-substs } (\text{fst } e) (\text{snd } e)))$ 
    by force
  qed
  show ?right  $\subseteq$  ?left proof
    fix  $x\ t$  assume  $(x,t) \in (\bigcup e \in \text{set } ds. \text{set } (\text{right-substs } (\text{fst } e) (\text{snd } e)))$ 
    then obtain  $j$  where  $j1:j < \text{length } ss$  and  $j2:j < \text{length } ts$  and  $(x,t) \in \text{set } (\text{right-substs } (ss!j) (ts!j))$ 
    unfolding  $ds$  by (metis (no-types, lifting) UN-E in-set-zip)

```

then obtain q where $1:q \in \text{fun-poss } (ss!j)$ and $2:(ss!j)|-q = t$ and $3:q \in \text{poss } (ts!j)$ and $4:(ts!j)|-q = \text{Var } x$
 by (meson right-substs-imp-props)
 then have $j\#q \in \text{fun-poss } (\text{Fun } f \text{ } ss)$ and $(\text{Fun } f \text{ } ss)|-(j\#q) = t$ and $(j\#q) \in \text{poss } (\text{Fun } g \text{ } ts)$ and $(\text{Fun } g \text{ } ts)|-(j\#q) = \text{Var } x$
 using $j1 \ j2$ by auto
 then show $(x,t) \in \text{set } (\text{right-substs } (\text{Fun } f \text{ } ss) (\text{Fun } g \text{ } ts))$
 by (meson props-imp-right-substs)
 qed
 qed

lemma subst-compose-id:
 assumes $\bigwedge \tau. \tau \in \text{set } \tau s \implies t \cdot \tau = t$
 shows $t \cdot (\text{compose } \tau s) = t$
 using assms by (induct τs) simp-all

lemma subst-compose-distinct-vars:
 assumes $\sigma = \text{compose } \tau s$ and $\text{part}:\text{is-partition } (\text{map vars-subst } \tau s)$
 and $\tau i:\tau i \in \text{set } \tau s$ and $s:\tau i \ x = s \ s \neq \text{Var } x$
 shows $\sigma \ x = s$
 proof-
 from τi obtain i where $i:i < \text{length } \tau s$ and $\tau s ! i = \tau i$
 by (metis in-set-idx)
 then have $\tau s:\tau s = (\text{take } i \ \tau s) @ \tau i \# (\text{drop } (\text{Suc } i) \ \tau s)$
 using id-take-nth-drop by blast
 from s have $x\text{-vars-subst}:x \in \text{vars-subst } \tau i$
 by (metis fun-upd-same fun-upd-triv subst-apply-term-empty subst-compose
 vars-subst-compose-update)
 {fix j assume $j < i$
 with $\text{part } i \ x\text{-vars-subst}$ have $x \notin \text{vars-subst } (\tau s ! j)$
 unfolding is-partition-alt is-partition-alt-def
 by (metis (no-types, lifting) Int-iff dual-order.strict-trans equals0D is-partition-def
 length-map nth-map part)
 then have $(\tau s ! j) \ x = \text{Var } x$
 unfolding vars-subst-def by (meson UnI1 notin-subst-domain-imp-Var)
 }
 then have $\text{take-}i\text{-}\tau s:\text{compose } (\text{take } i \ \tau s) \ x = \text{Var } x$
 using subst-compose-id[of take $i \ \tau s \ \text{Var } x$] using in-set-idx by force
 {fix y assume $y \in \text{vars-term } s$
 with s have $y\text{-vars-subst}:y \in \text{vars-subst } \tau i$
 unfolding vars-subst-def by (metis UnI2 Union-iff image-eqI notin-subst-domain-imp-Var
 subst-range.simps)
 {fix j assume $i < j \ j < \text{length } \tau s$
 with $\text{part } i \ y\text{-vars-subst}$ have $y \notin \text{vars-subst } (\tau s ! j)$
 unfolding is-partition-alt is-partition-alt-def
 by (metis (no-types, lifting) Int-iff equals0D is-partition-def length-map
 nth-map part)
 then have $(\tau s ! j) \ y = \text{Var } y$
 unfolding vars-subst-def by (meson UnI1 notin-subst-domain-imp-Var)
 }

```

    }
    then have compose (drop (Suc i)  $\tau s$ )  $y = \text{Var } y$ 
      using subst-compose-id[of drop (Suc i)  $\tau s$   $\text{Var } y$ ] using in-set-idx by force
  }
  then have  $s \cdot (\text{compose } (\text{drop } (\text{Suc } i) \tau s)) = s$ 
    by (simp add: term-subst-eq)
  with take-i- $\tau s$  s(1) i show ?thesis
    by (metis  $\tau s$  assms(1) compose-append compose-simps(3) eval-term.simps(1)
    subst-compose)
qed

```

lemma *subst-id-compose*:

```

  assumes  $\sigma = \text{compose } \tau s$  and part:is-partition (map vars-subst  $\tau s$ )
    and  $t \cdot \sigma = t$ 
    and  $\tau \in \text{set } \tau s$ 
  shows  $t \cdot \tau = t$ 
  using assms subst-compose-distinct-vars by (metis (full-types) subst-apply-term-empty
  term-subst-eq-conv)

```

lemma *compose-subst-of*:

```

  assumes set ss =  $\bigcup$  (set 'set ss')
    and is-partition (map (vars-term  $\circ$  snd) ss) and distinct (map fst ss)
    and set (map fst ss)  $\cap \bigcup$  (set (map (vars-term  $\circ$  snd) ss)) = {}
    and is-partition (map vars-subst (map subst-of ss'))
  shows subst-of ss = compose (map subst-of ss') (is ? $\sigma = ?\tau$ )
proof
  fix x
  show ? $\sigma$  x = ? $\tau$  x proof(cases x  $\in$  set (map fst ss))
    case True
    then obtain s where s:(x, s)  $\in$  set ss
      by fastforce
    then have  $\sigma$ -x:? $\sigma$  x = s
      using assms(3) by (smt (verit) UN-I assms(4) case-prodI2 disjoint-iff
      eq-key-imp-eq-value list.set-map o-apply prod.sel(2) subst-of-apply)
    from s have s-x:s  $\neq$  Var x
      using assms(4) by fastforce
    from s obtain ssi where ssi:(x, s)  $\in$  set ssi ssi  $\in$  set ss'
      using assms(1) by auto
    then have subst-of ssi x = s
      using assms(1,3,4) by (smt (verit, ccfv-threshold) UN-I case-prodI2 disjoint-iff
      eq-key-imp-eq-value image-iff list.set-map o-apply snd-conv subst-of-apply)
    with assms(5) have ? $\tau$  x = s
      using subst-compose-distinct-vars ssi(2) s-x by (smt (verit, del-insts) in-set-idx
      length-map nth-map nth-mem)
    with  $\sigma$ -x show ?thesis by simp
  next
  case False
  then have  $\sigma$ -x:? $\sigma$  x = Var x
    by (simp add: not-elem-subst-of)

```

```

{fix ssi assume ssi ∈ set ss'
  with False assms(1) have x ∉ set (map fst ssi)
  by auto
  then have (subst-of ssi) x = Var x
  by (simp add: not-elem-subst-of)
}
then have ?τ x = Var x
  using subst-compose-id by (smt (verit, ccfv-SIG) eval-term.simps(1) image-iff
list.set-map)
  with σ·x show ?thesis by simp
qed
qed

```

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) ∩ vars-term (Fun g ts) = {}
  and decompose (Fun f ss) (Fun g ts) = Some ds
  and i:i < length ds and σ:σ = linear-unifier (fst (ds!i)) (snd (ds!i))
  and j:j < length ds j ≠ i
shows fst (ds!j) · σ = fst (ds!j) ∧ snd (ds!j) · σ = snd (ds!j)
proof-
  from assms have zip:ds = zip ss ts and l:length ss = length ts
  using decompose-Some by blast+
  from i j obtain si ti where s-t-i:ds ! i = (si, ti) ss ! i = si ts ! i = ti
  using l zip by force
  from j obtain sj tj where s-t-j:ds ! j = (sj, tj) ss ! j = sj ts ! j = tj
  using l zip by force
  have vars-term sj ∩ vars-term ti = {}
  using i j s-t-i s-t-j disj l zip by fastforce
  moreover have vars-term sj ∩ vars-term si = {}
  using lin(1) s-t-i s-t-j i j var-in-linear-args
  by (metis Int-emptyI l length-map map-fst-zip zip)
  moreover have vars-term tj ∩ vars-term ti = {}
  using lin(2) s-t-i s-t-j i j l var-in-linear-args
  by (metis Int-emptyI l length-map map-fst-zip zip)
  moreover have vars-term tj ∩ vars-term si = {}
  using i j s-t-i s-t-j disj l zip by fastforce
  moreover from σ s-t-i have vars-subst σ ⊆ vars-term si ∪ vars-term ti
  by (metis fst-conv snd-conv vars-subst-linear-unifier)
  ultimately show ?thesis
  unfolding s-t-i s-t-j fst-conv snd-conv
  by (metis inf-sup-distrib1 subst-apply-term-ident sup.absorb-iff2 sup-bot.neutr-eq-iff
vars-subst-def)
qed

```

lemma linear-unifier-decompose:

```

assumes linear-term (Fun f ss) linear-term (Fun g ts)
  and disj:vars-term (Fun f ss) ∩ 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 ($\lambda(s,t). \text{linear-unifier } s \ t$) ds)

proof –

let ?ls=left-substs (Fun f ss) (Fun g ts) and ?rs=right-substs (Fun f ss) (Fun g ts)

have left:set ?ls = ($\bigcup (s, t) \in \text{set } ds. \text{set } (\text{left-substs } s \ t)$)

using decompose-left-substs[OF ds] by auto

have right:set ?rs = ($\bigcup (s, t) \in \text{set } ds. \text{set } (\text{right-substs } s \ t)$)

using decompose-right-substs[OF ds] by auto

from left right have sets:set (?ls @ ?rs) = $\bigcup (\text{set } ' \text{set } (\text{map } (\lambda(s, t). \text{left-substs } s \ t \ @ \ \text{right-substs } s \ t) \ ds))$

by auto

{fix l assume l \in (set (map (vars-term \circ snd) ?ls))

then obtain t' where t' \in set (map snd ?ls) and vars-term t' = l

by auto

then have l \subseteq vars-term (Fun g ts)

using map-snd-left-substs by blast

}

then have 1: $\bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?ls)) \subseteq \text{vars-term } (\text{Fun } g \ ts)$

using Union-least by blast

{fix r assume r \in (set (map (vars-term \circ snd) ?rs))

then obtain t' where t' \in set (map snd ?rs) and vars-term t' = r

by auto

then have r \subseteq vars-term (Fun f ss)

using map-snd-right-substs by blast

}

then have 2: $\bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?rs)) \subseteq \text{vars-term } (\text{Fun } f \ ss)$

using Union-least by blast

have snd-disj: $\bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?ls)) \cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?rs)) = \{\}$

using 1 2 assms(3) by blast

then have part:is-partition (map (vars-term \circ snd) (?ls @ ?rs))

using is-partition-append[OF is-partition-map-snd-left-substs[OF assms(2,1)] is-partition-map-snd-right-substs[OF assms(2,1)]]

unfolding length-map map-append by (simp add: Union-disjoint)

have dist:distinct (map fst (?ls @ ?rs))

using distinct-append distinct-map-fst-left-substs[OF assms(1)] distinct-map-fst-right-substs[OF assms(2)] map-fst-left-substs map-fst-right-substs

by (smt (verit, del-insts) disj inf.orderE inf-assoc inf-bot-right inf-left-commute map-append)

have set (map fst ?ls) $\cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?ls)) = \{\}$

by (meson 1 disj disjoint-iff map-fst-left-substs subsetD)

moreover have set (map fst ?ls) $\cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?rs)) = \{\}$

using assms(1) distinct-fst-lsubsts-snd-rsubsts by blast

moreover have set (map fst ?rs) $\cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?rs)) = \{\}$

by (meson 2 disj disjoint-iff map-fst-right-substs subsetD)

moreover have set (map fst ?rs) $\cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?ls)) = \{\}$

using assms(2) distinct-fst-rsubsts-snd-lsubsts by blast

ultimately have disj:set (map fst (?ls @ ?rs)) $\cap \bigcup (\text{set } (\text{map } (\text{vars-term } \circ \text{snd}) \ ?rs)) = \{\}$

```

(?ls @ ?rs)) = {}
  unfolding map-append set-append by (simp add: boolean-algebra.conj-disj-distrib
boolean-algebra.conj-disj-distrib2)
  have part2:is-partition (map vars-subst (map subst-of (map ( $\lambda(s, t).$  left-substs
s t @ right-substs s t) ds)))
  using decompose-is-partition-vars-subst[OF assms(1,2,3,4)]
  by (metis (mono-tags, lifting) case-prod-beta length-map map-nth-eq-conv)
  show ?thesis using compose-subst-of[OF sets part dist disj part2]
  by (smt (verit, del-insts) case-prod-unfold length-map map-nth-eq-conv)
qed

```

lemma unify-linear-terms:

```

assumes unify es substs = Some res
  and compose (subst-of substs # (map ( $\lambda(s,t).$  linear-unifier s t) es)) =  $\tau$ 
  and  $\forall t \in \text{set } (\text{map fst } \text{es}) \cup \text{set } (\text{map snd } \text{es}). \text{linear-term } t$ 
  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 =  $\tau$ 
  using assms proof(induct arbitrary: res substs  $\tau$  rule:unify.induct)
  case (2 f ss g ts E)
  from 2(2) obtain ds where ds':decompose (Fun f ss) (Fun g ts) = Some ds
  unfolding unify.simps by fastforce
  then have ds:ds = zip ss ts and l:length ss = length ts
  by fastforce+
  with 2(4) have  $\forall t \in \text{set } (\text{map fst } \text{ds}). \text{linear-term } t$ 
  using map-fst-zip by (metis (no-types, lifting) UnCI fst-conv linear-term.simps(2)
list.set-intros(1) list.simps(9))
  moreover from 2(4) ds l have  $\forall t \in \text{set } (\text{map snd } \text{ds}). \text{linear-term } t$ 
  using map-snd-zip by (metis (no-types, lifting) UnCI linear-term.simps(2)
list.set-intros(1) list.simps(9) snd-conv)
  ultimately have lin: $\forall a \in \text{set } (\text{map fst } (\text{ds} @ E)) \cup \text{set } (\text{map snd } (\text{ds} @ E)).$ 
linear-term a
  using 2(4) by (metis UnE UnI1 UnI2 list.set-intros(2) list.simps(9) map-append
set-append)
  have lin-f-g:linear-term (Fun f ss) linear-term (Fun g ts)
  using 2(4) by auto
  from 2(6) have vars:vars-term (Fun f ss)  $\cap$  vars-term (Fun g ts) = {}
  by fastforce
  from ds' 2(2) have unif:unify (ds @ E) substs = Some res
  by auto
  have compose (map ( $\lambda a. \text{case } a \text{ of } (s, t) \Rightarrow \text{linear-unifier } s t$ ) ds) = linear-unifier
(Fun f ss) (Fun g ts)
  using linear-unifier-decompose[OF lin-f-g vars ds] by fastforce
  then have  $\tau$ 2:compose (subst-of substs # map ( $\lambda a. \text{case } a \text{ of } (s, t) \Rightarrow \text{lin-}$ 
ear-unifier s t) (ds @ E)) =  $\tau$ 
  using 2(3) compose-append by simp

```

```

{fix i j σ assume i:i < j and j:j < length (ds @ E) and σ:σ = linear-unifier
(fst ((ds @ E) ! i)) (snd ((ds @ E) ! i))
  have fst ((ds @ E) ! j) · σ = fst ((ds @ E) ! j) ∧ snd ((ds @ E) ! j) · σ = snd
((ds @ E) ! j)
  proof(cases i < length ds)
    case True
    then have σ:σ = linear-unifier (fst (ds ! i)) (snd (ds ! i))
      by (simp add: σ dual-order.strict-trans i nth-append)
    show ?thesis proof(cases j < length ds)
      case True
      have lin:linear-term (Fun f ss) linear-term (Fun g ts)
        using 2(4) by simp+
      show ?thesis
        using linear-term-decompose-subst-id[OF lin vars ds' ⟨i < length ds⟩ σ
True] i True
        by (simp add: j nat-neq-iff nth-append)
      next
      case False
      let ?j'=j - length ds
      let ?τ=linear-unifier (Fun f ss) (Fun g ts)
      from False j have ?j' < length E
        by fastforce
      then have fst:fst (E ! ?j') · ?τ = fst (E ! ?j') and snd:snd (E ! ?j') · ?τ
= snd (E ! ?j')
        using 2(5) by force+
      have fst (E ! ?j') · σ = fst (E ! ?j')
        using subst-id-compose[OF linear-unifier-decompose[OF lin-f-g vars ds']
decompose-is-partition-vars-subst[OF lin-f-g vars ds']]
      by (smt (verit, best) True σ ds fst in-set-conv-nth l length-map map2-map-map
map-fst-zip map-snd-zip nth-map)
      moreover have snd (E ! ?j') · σ = snd (E ! ?j')
        using subst-id-compose[OF linear-unifier-decompose[OF lin-f-g vars ds']
decompose-is-partition-vars-subst[OF lin-f-g vars ds']]
      by (smt (verit, best) True σ ds snd in-set-conv-nth l length-map map2-map-map
map-fst-zip map-snd-zip nth-map)
      ultimately show ?thesis
        by (simp add: False nth-append)
    qed
  next
  case False
  let ?i'=i - length ds
  have i':?i' < length E
    using False i j by force
  from σ have σ':σ = linear-unifier (fst (E ! ?i')) (snd (E ! ?i'))
    by (simp add: False nth-append)
  let ?j'=j - length ds
  from False i j have ?i' < ?j'
    by simp
  moreover with j have ?j' < length E

```

```

      by fastforce
      ultimately show ?thesis
        using 2(5) i' σ' by (smt (verit, best) length-nth-simps(2) nat-diff-split
not-less-eq not-less-zero nth-Cons-Suc nth-append)
      qed
    }
  moreover
  { fix i assume i:i < length (ds @ E)
    have vars-term (fst ((ds @ E) ! i)) ∩ vars-term (snd ((ds @ E) ! i)) = {}
    proof (cases i < length ds)
      case True
        with ds have vars-term (fst (ds!i)) ⊆ vars-term (Fun f ss)
          using nth-mem by auto
        moreover from True ds have vars-term (snd (ds!i)) ⊆ vars-term (Fun g ts)
          using nth-mem by auto
        ultimately show ?thesis
          using 2(6) True by (metis Int-mono bot.extremum-uniqueI nth-append vars)
      next
      case False
        let ?i'=i - length ds
        have i':?i' < length E
          using False i by force
        with 2(6) have vars-term (fst (E ! ?i')) ∩ vars-term (snd (E ! ?i')) = {}
          by force
        then show ?thesis
          by (simp add: False nth-append)
      qed
    }
  ultimately show ?case
    using 2(1)[OF ds' unif τ2 lin] by blast
next
case (3 x t E)
show ?case proof (cases t = Var x)
  case True
    from 3(3) have unif:unify E substs = Some res
      unfolding True unify.simps by simp
    from 3(4) have τ2:compose (subst-of substs # map (λa. case a of (s, t) ⇒
linear-unifier s t) E) = τ
      unfolding True append-Cons list.map compose-simps using linear-unifier-same
by (metis Var-subst-compose old.prod.case)
    from 3(5) have lin:∀ a∈set (map fst E) ∪ set (map snd E). linear-term a
      by simp
    from 3(6) have ∧i σ. i < length E ⇒ σ = linear-unifier (fst (E ! i)) (snd
(E ! i)) ⇒
      (∀ j<length E. i < j ⇒ fst (E ! j) · σ = fst (E ! j) ∧ snd (E ! j) · σ =
snd (E ! j))
      by (metis length-nth-simps(2) not-less-eq nth-Cons-Suc)
    moreover have (∧i. i < length E ⇒ vars-term (fst (E ! i)) ∩ vars-term (snd
(E ! i)) = {})

```

```

    using 3(7) by fastforce
    ultimately show ?thesis using 3(1)[OF True unif  $\tau 2$  lin] by simp
next
  case False
  with 3(3) have  $x:x \notin \text{vars-term } t$ 
    by fastforce
  with 3(3) False have  $\text{unif:unify (subst-list (subst x t) E) ((x, t) \# \text{substs}) =$ 
    Some res
    by simp
  let  $\sigma = (\text{subst } x \ t)$ 
  have  $\sigma:\text{linear-unifier (Var } x) \ t = ?\sigma$ 
    using linear-unifier-var1 by simp
  from 3(7) have  $\text{subst-list:subst-list (subst x t) } E = E$ 
  proof-
    {fix j assume  $j < \text{length } E$ 
      then have  $j:\text{Suc } j < \text{length } ((\text{Var } x, t) \# E)$ 
        by simp
      with 3(6)[of 0 Suc j  $\sigma$ ]  $\sigma$  have  $\text{fst } (E ! j) \cdot ?\sigma = \text{fst } (E ! j) \wedge \text{snd } (E ! j)$ 
         $\cdot ?\sigma = \text{snd } (E ! j)$ 
        by (metis fst-conv length-nth-simps(2) nth-Cons-0 nth-Cons-Suc snd-conv
          zero-less-Suc)
    }
    then show ?thesis
      unfolding subst-list-def by (simp add: map-nth-eq-conv)
  qed
  have  $\tau 2:\text{compose (subst-of } ((x, t) \# \text{substs}) \# \text{map } (\lambda a. \text{case } a \text{ of } (s, t) \Rightarrow$ 
    linear-unifier s t) (subst-list (subst x t) E)) =  $\tau$ 
    using 3(4) unfolding subst-list list.map prod.case  $\sigma$  subst-of-simps(3) com-
    pose-append fst-conv snd-conv compose-simps(1,3)
    using subst-compose-assoc by blast
  from 3(5) have  $\text{lin}:\forall a \in \text{set (map fst (subst-list (subst x t) E))} \cup \text{set (map snd$ 
    (subst-list (subst x t) E)). linear-term a
    unfolding subst-list by simp
    {fix i j  $\sigma$  assume  $i:i < j$  and  $j:j < \text{length } E$  and  $\sigma'':\sigma = \text{linear-unifier (fst$ 
      (E ! i)) (snd (E ! i))
      with 3(6) have  $1:\text{fst } (E ! j) \cdot \sigma = \text{fst } (E ! j) \wedge \text{snd } (E ! j) \cdot \sigma = \text{snd } (E ! j)$ 
        by (metis length-nth-simps(2) not-less-eq nth-Cons-Suc)
    }
    moreover have  $(\bigwedge i. i < \text{length } E \Rightarrow \text{vars-term (fst } (E ! i)) \cap \text{vars-term (snd$ 
      (E ! i)) =  $\{\}$ )
      using 3(7) by fastforce
    ultimately show ?thesis
      using 3(2)[OF False x unif  $\tau 2$  lin] 3(7) unfolding subst-list subst-of-simps(3)
  by simp
qed
next
  case (4 f ts x E)
  from 4(2) have  $x:x \notin \text{vars-term (Fun f ts)}$ 
    by fastforce

```

```

with 4(2) have unif:unify (subst-list (subst x (Fun f ts)) E) ((x, Fun f ts) #
substs) = Some res
  by auto
  let ? $\sigma$ =(subst x (Fun f ts))
  have  $\sigma$ :linear-unifier (Fun f ts) (Var x) = ? $\sigma$ 
    using linear-unifier-var2 by simp
  have subst-list:subst-list (subst x (Fun f ts)) E = E
  proof–
    {fix j assume j < length E
      then have Suc j < length ((Fun f ts, Var x) # E)
        by simp
      with 4(5)  $\sigma$  have fst (E ! j) · ? $\sigma$  = fst (E ! j) ∧ snd (E ! j) · ? $\sigma$  = snd (E
! j)
        by (metis fst-conv length-nth-simps(2) nth-Cons-0 nth-Cons-Suc snd-conv
zero-less-Suc)
      }
    then show ?thesis
      unfolding subst-list-def by (simp add: map-nth-eq-conv)
    qed
    have  $\tau$ 2:compose (subst-of ((x, Fun f ts) # substs) # map ( $\lambda a$ . case a of (s, t)
⇒ linear-unifier s t) (subst-list (subst x (Fun f ts)) E)) =  $\tau$ 
      using 4(3) unfolding subst-list list.map prod.case  $\sigma$  subst-of-simps(3) com-
pose-append fst-conv snd-conv compose-simps(1,3) by (simp add: subst-compose-assoc)
      from 4(4) have lin: $\forall a \in \text{set } (\text{map } \text{fst } (\text{subst-list } (\text{subst } x \text{ (Fun } f \text{ ts)) } E)) \cup \text{set}$ 
(map snd (subst-list (subst x (Fun f ts)) E)). linear-term a
        unfolding subst-list by simp
        {fix i j  $\sigma$  assume i:i < j and j:j < length E and  $\sigma''$ : $\sigma$  = linear-unifier (fst (E
! i)) (snd (E ! i))
          with 4(5) have 1:fst (E ! j) ·  $\sigma$  = fst (E ! j) ∧ snd (E ! j) ·  $\sigma$  = snd (E ! j)
            by (metis length-nth-simps(2) not-less-eq nth-Cons-Suc)
          }
        moreover have ( $\bigwedge i$ . i < length E ⇒ vars-term (fst (E ! i)) ∩ vars-term (snd
(E ! i)) = {})
          using 4(6) by fastforce
          ultimately show ?case
          using 4(1)[OF x unif  $\tau$ 2 lin] 4(6) unfolding subst-list by simp
        qed auto

```

lemma *mgu-distinct-vars-term-list*:

```

assumes unif:unifiers {(s, t)} ≠ {}
and distinct:distinct ((vars-term-list s) @ (vars-term-list t))
shows mgu s t = Some (linear-unifier s t)

```

proof–

```

let ? $\tau$ u=linear-unifier s t
from unif have mgu s t ≠ None
  by (meson mgu-complete)
then obtain us where us:unify [(s, t)] [] = Some us
  unfolding mgu-def by fastforce
have tau:compose (subst-of [] # map ( $\lambda(s, t)$ . linear-unifier s t) [(s, t)] = ? $\tau$ u

```

```

    by simp
  have lin:  $\forall t \in \text{set } (\text{map fst } [(s, t)]) \cup \text{set } (\text{map snd } [(s, t)]).$  linear-term  $t$ 
    using distinct distinct-vars-linear-term by auto
  have vars-term  $s \cap \text{vars-term } t = \{\}$ 
    using distinct by simp
  then have subst-of  $us = ?\tau$ 
    using unify-linear-terms[OF  $us \ \tau \ \text{lin}$ ] by simp
  then show ?thesis
    using  $us$  by (simp add: mgu-def)
qed

end

```

15 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* $\sigma \ E$

using *assms* **by** (*fastforce simp: is-mgu-def unifiers-def*)

lemma *subst-set-insert* [*simp*]:

subst-set $\sigma \ (\text{insert } e \ E) = \text{insert } (\text{fst } e \cdot \sigma, \text{snd } e \cdot \sigma) \ (\text{subst-set } \sigma \ E)$

by (*auto simp: subst-set-def*)

lemma *unifiable-UnD* [*dest*]:

unifiable $(M \cup N) \implies \text{unifiable } M \wedge \text{unifiable } N$

by (*auto simp: unifiable-def*)

lemma *supt-imp-not-unifiable*:

assumes $s \triangleright t$

shows $\neg \text{unifiable } \{(t, s)\}$

proof

assume *unifiable* $\{(t, s)\}$

then obtain σ **where** $\sigma \in \text{unifiers } \{(t, s)\}$

by (*auto simp: unifiable-def*)

then have $t \cdot \sigma = s \cdot \sigma$ **by** (*auto*)

moreover have $s \cdot \sigma \triangleright t \cdot \sigma$

using *assms* **by** (*metis instance-no-supt-imp-no-supt*)

ultimately show *False* **by** *auto*

qed

```

lemma unifiable-insert-Var-swap [simp]:
  unifiable (insert (t, Var x) E)  $\longleftrightarrow$  unifiable (insert (Var x, t) E)
by (rule unifiable-insert-swap)

lemma unifiers-Int1 [simp]:
  (s, t)  $\in$  E  $\implies$  unifiers {(s, t)}  $\cap$  unifiers E = unifiers E
by (auto simp: unifiers-def)

lemma imgu-linear-var-disjoint:
  assumes is-imgu  $\sigma$  {(l2 |- p, l1)}
  and  $p \in \text{poss } l2$ 
  and linear-term l2
  and vars-term l1  $\cap$  vars-term l2 = {}
  and  $q \in \text{poss } l2$ 
  and parallel-pos p q
  shows  $l2 \vdash q = l2 \vdash q \cdot \sigma$ 
using assms
proof (induct p arbitrary: q l2)
  case (Cons i p)
  from this(3) obtain f ls where
    l2[simp]:  $l2 = \text{Fun } f \text{ } ls$  and
    i:  $i < \text{length } ls$  and
    p:  $p \in \text{poss } (ls ! i)$ 
    by (cases l2) (auto)
  then have l2i:  $l2 \vdash ((i \# p)) = ls ! i \vdash p$  by auto
  have linear-term (ls ! i) using Cons(4) l2 i by simp
  moreover have vars-term l1  $\cap$  vars-term (ls ! i) = {} using Cons(5) l2 i by
  force
  ultimately have IH:  $\bigwedge q. q \in \text{poss } (ls ! i) \implies p \perp q \implies ls ! i \vdash q = ls ! i \vdash q \cdot \sigma$ 
  using Cons(1)[OF Cons(2)[unfolded l2i] p] by blast
  from Cons(7) obtain j q' where  $q = j \# q'$  by (cases q) auto
  show ?case
  proof (cases j = i)
  case True with Cons(6,7) IH q show ?thesis by simp
  next
  case False
  from Cons(6) q have  $j < \text{length } ls$  by simp
  { fix y
    assume  $y \in \text{vars-term } (l2 \vdash q)$ 
    let  $? \tau = \lambda x. \text{if } x = y \text{ then Var } y \text{ else } \sigma x$ 
    from y Cons(6) q j have  $yj: y \in \text{vars-term } (ls ! j)$ 
    by simp (meson subt-at-imp-supteq subteq-Var-imp-in-vars-term supteq-Var
    supteq-trans)
    { fix i j
      assume  $j < \text{length } ls$  and  $i < \text{length } ls$  and  $\text{neg: } i \neq j$ 
      from j Cons(4) have  $\forall i < j. \text{vars-term } (ls ! i) \cap \text{vars-term } (ls ! j) = \{\}$ 
      by (auto simp: is-partition-def)
    }
  }

```

```

    moreover from  $i \text{ Cons}(4)$  have  $\forall j < i. \text{vars-term } (ls ! i) \cap \text{vars-term } (ls ! j) = \{\}$ 
    by (auto simp : is-partition-def)
    ultimately have  $\text{vars-term } (ls ! i) \cap \text{vars-term } (ls ! j) = \{\}$ 
    using neq by (cases  $i < j$ ) auto
  }
  from this[OF  $i j \text{ False}$ ] have  $y \notin \text{vars-term } (ls ! i)$  using  $yj$  by auto
  then have  $y \notin \text{vars-term } (l2 \mid - ((i \# p)))$ 
  by (metis  $l2i \ p \ \text{subt-at-imp-supteq} \ \text{subteq-Var-imp-in-vars-term} \ \text{supteq-Var-supteq-trans}$ )
  then have  $\forall x \in \text{vars-term } (l2 \mid - ((i \# p))). \ ?\tau \ x = \sigma \ x$  by auto
  then have  $l2\tau\sigma: l2 \mid - ((i \# p)) \cdot ?\tau = l2 \mid - ((i \# p)) \cdot \sigma$  using  $\text{term-subst-eq}[of \ - \ \sigma \ ?\tau]$  by simp
  from  $\text{Cons}(5)$  have  $y \notin \text{vars-term } l1$  using  $y \text{ Cons}(6) \ \text{vars-term-subt-at}$  by fastforce
  then have  $\forall x \in \text{vars-term } l1. \ ?\tau \ x = \sigma \ x$  by auto
  then have  $l1\tau\sigma: l1 \cdot ?\tau = l1 \cdot \sigma$  using  $\text{term-subst-eq}[of \ - \ \sigma \ ?\tau]$  by simp
  have  $l1 \cdot \sigma = l2 \mid - (i \# p) \cdot \sigma$  using  $\text{Cons}(2) \ \text{unfolding} \ \text{is-ingu-def}$  by auto
  then have  $l1 \cdot ?\tau = l2 \mid - (i \# p) \cdot ?\tau$  using  $l1\tau\sigma \ l2\tau\sigma$  by simp
  then have  $?\tau \in \text{unifiers } \{(l2 \mid - (i \# p), l1)\}$  unfolding  $\text{unifiers-def}$  by simp
  with  $\text{Cons}(2)$  have  $\tau\sigma: ?\tau = \sigma \circ_s ?\tau$  unfolding  $\text{is-ingu-def}$  by blast
  have  $\text{Var } y = \text{Var } y \cdot \sigma$ 
  proof (rule ccontr)
    let  $?x = \text{Var } y \cdot \sigma$ 
    assume *:  $\text{Var } y \neq ?x$ 
    have  $\text{Var } y = \text{Var } y \cdot ?\tau$  by auto
    also have  $\dots = (\text{Var } y \cdot \sigma) \cdot ?\tau$  using  $\tau\sigma \ \text{subst-subst}$  by metis
    finally have  $xy: ?x \cdot \sigma = \text{Var } y$  using * by (cases  $\sigma \ y$ ) auto
    have  $\sigma \circ_s \sigma = \sigma$  using  $\text{Cons}(2) \ \text{unfolding} \ \text{is-ingu-def}$  by auto
    then have  $?x \cdot (\sigma \circ_s \sigma) = \text{Var } y$  using  $xy$  by auto
    moreover have  $?x \cdot \sigma \cdot \sigma = ?x$  using  $xy$  by auto
    ultimately show  $\text{False}$  using * by auto
  qed
}
then show  $?thesis$  by (simp add:  $\text{term-subst-eq}$ )
qed
qed auto
end

```

16 Renaming of Terms, Substitutions, TRSs, ...

theory *Renaming-Interpretations*

imports

Renaming

Trs

Unification-More

Unifiers-More

begin

lemma *variants-imp-bij-betw-vars*:

assumes $s \cdot \sigma = t$ and $t \cdot \tau = s$

shows *bij-betw* (*the-Var* $\circ \sigma$) (*vars-term* s) (*vars-term* t)

proof –

have *id*: (*the-Var* $\circ \sigma$) ‘ *vars-term* $s = \text{vars-term } t$

using *variants-imp-image-vars-term-eq* [*OF* *assms*] **by** *simp*

then have $\text{card } (\text{vars-term } t) \leq \text{card } (\text{vars-term } s)$

by (*metis card-image-le finite-vars-term*)

moreover have (*the-Var* $\circ \tau$) ‘ *vars-term* $t = \text{vars-term } s$

using *variants-imp-image-vars-term-eq* [*OF* *assms*(2, 1)] **by** *simp*

ultimately have $\text{card } (\text{vars-term } t) = \text{card } (\text{vars-term } s)$

by (*metis card-image-le eq-iff finite-vars-term*)

then have $\text{card } ((\text{the-Var } \circ \sigma) \text{ ‘ vars-term } s) = \text{card } (\text{vars-term } s)$

using *id* **by** *auto*

from *finite-card-eq-imp-bij-betw* [*OF* - *this*] *id*

show *?thesis* **by** *auto*

qed

When two terms are substitution instances of each other, then there is a variable renaming (with finite domain) between them.

lemma *variants-imp-renaming*:

fixes $s \ t :: ('f, 'v) \text{ term}$

assumes $s \cdot \sigma = t$ and $t \cdot \tau = s$

shows $\exists f. \text{bij } f \wedge \text{finite } \{x. f \ x \neq x\} \wedge s \cdot (\text{Var } \circ f) = t$

proof –

from *variants-imp-bij-betw-vars* [*OF* *assms*, *THEN* *bij-betw-extend*, of *UNIV*]

obtain *g* where *: $\forall x \in \text{vars-term } s. g \ x = (\text{the-Var } \circ \sigma) \ x$

and $\text{finite } \{x. g \ x \neq x\}$

and *bij* *g* **by** *auto*

moreover have 1: $\forall x \in \text{vars-term } s. (\text{Var } \circ g) \ x = \sigma \ x$

proof

fix *x*

assume $x \in \text{vars-term } s$

with * have $g \ x = (\text{the-Var } \circ \sigma) \ x$ **by** *simp*

with *variants-imp-is-Var* [*OF* *assms*] and $\langle x \in \text{vars-term } s \rangle$

show $(\text{Var } \circ g) \ x = \sigma \ x$ **by** *simp*

qed

moreover have $s \cdot (\text{Var } \circ g) = t$

using 1 $\langle s \cdot \sigma = t \rangle$ **by** (*auto simp: term-subst-eq-conv*)

ultimately show *?thesis* **by** *blast*

qed

Turning a permutation into a substitution.

abbreviation $\text{sop } \pi \equiv \text{Var } \circ \text{Rep-perm } \pi$

fun *permute-term* :: $'v \text{ perm} \Rightarrow ('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term}$

where

$\text{permute-term } p \text{ (Var } x) = \text{Var (permute-atom } p \text{ } x) \mid$
 $\text{permute-term } p \text{ (Fun } f \text{ } ts) = \text{Fun } f \text{ (map (permute-term } p) \text{ } ts)$

interpretation *term-pt: permutation-type permute-term*
apply *unfold-locales*
subgoal for *x* **by** (*induct x, auto simp: map-idI*)
subgoal for *p q x* **by** (*induct x, auto simp: map-idI*)
done

adhoc-overloading
 $\text{PERMUTE permute-term and}$
 $\text{FRESH term-pt.fresh term-pt.fresh-set}$

interpretation *terms-pt: list-pt permute-term ..*

adhoc-overloading
 $\text{PERMUTE terms-pt.permute-list and}$
 $\text{FRESH terms-pt.fresh}$

lemma *supp-Var:*
fixes *x :: 'v :: infinite*
shows $\text{term-pt.supp (Var } x) = \{x\}$
using *infinite-UNIV*
by (*auto simp: term-pt.supp-def swap-atom*)

lemma *permute-Fun:*
fixes *p :: ('v :: infinite) perm*
shows $p \cdot (\text{Fun } f \text{ } ss) = \text{Fun } f \text{ (} p \cdot ss \text{)}$
by (*induct ss (simp)+*)

lemma *supp-Fun':*
 $\text{term-pt.supp (Fun } f \text{ } ss) = \text{permutation-type.supp } (\cdot) \text{ } ss$
by (*simp only: term-pt.supp-def terms-pt.supp-def permute-Fun auto*)

lemma *supp-Fun:*
 $\text{term-pt.supp (Fun } f \text{ } ss) = (\bigcup s \in \text{set } ss. \text{term-pt.supp } s)$
by (*simp add: supp-Fun'*)

lemma *supp-vars-term-eq:*
fixes *t :: ('f, 'v :: infinite) term*
shows $\text{term-pt.supp } t = \text{vars-term } t$
by (*induct t (simp add: supp-Var supp-Fun)+*)

interpretation *subst-conjugate-pt: fun-pt permute-atom permute-term ..*

abbreviation $\text{conjugate-subst} \equiv \text{subst-conjugate-pt.permute-fun}$

lemma *term-apply-subst-eqv [eqvt]:*

$p \cdot (t \cdot \sigma) = (p \cdot t) \cdot (\text{conjugate-subst } p \sigma)$
by (*induct* t) (*simp* *add*: *subst-conjugate-pt.permute-fun-def*)+

definition *permute-subst* :: ($'v :: \text{infinite}$) *perm* \Rightarrow ($'f, 'v$) *subst* \Rightarrow ($'f, 'v$) *subst*
where

$\text{permute-subst } \pi \sigma x = \pi \cdot \sigma x$

interpretation *subst-pt*: *permutation-type permute-subst*
by *standard* (*simp* *add*: *permute-subst-def [abs-def]*)+

adhoc-overloading

$\text{PERMUTE } \text{permute-subst}$ **and**
 $\text{FRESH } \text{subst-pt.fresh}$

fun *permute-ctxt* :: ($'v$ *perm* \Rightarrow ($'f, 'v$) *ctxt* \Rightarrow ($'f, 'v$) *ctxt*)
where

$\text{permute-ctxt } p \square = \square \mid$
 $\text{permute-ctxt } p (\text{More } f \text{ ss1 } C \text{ ss2}) =$
 $\text{More } f (\text{map } (\text{permute-term } p) \text{ ss1}) (\text{permute-ctxt } p C) (\text{map } (\text{permute-term } p) \text{ ss2})$

interpretation *ctxt-pt*: *permutation-type permute-ctxt*

apply *unfold-locales*
subgoal for x **by** (*induct* x , *auto simp*: *map-idI*)
subgoal for $p \ q \ x$ **by** (*induct* x , *auto simp*: *map-idI*)
done

adhoc-overloading

$\text{PERMUTE } \text{permute-ctxt}$ **and**
 $\text{FRESH } \text{ctxt-pt.fresh}$

lemma *supp-Hole*:

$\text{ctxt-pt.supp } \square = \{\}$
by (*simp* *add*: *ctxt-pt.supp-def*)

lemma *permute-More*:

fixes $p :: ('v :: \text{infinite}) \text{ perm}$
shows $p \cdot (\text{More } f \text{ ss1 } C \text{ ss2}) = \text{More } f (p \cdot \text{ss1}) (p \cdot C) (p \cdot \text{ss2})$
by (*simp*)

lemma *supp-More'*:

$\text{ctxt-pt.supp } (\text{More } f \text{ ss1 } C \text{ ss2}) =$
 $\text{permutation-type.supp } (\cdot) \text{ ss1} \cup \text{ctxt-pt.supp } C \cup \text{permutation-type.supp } (\cdot) \text{ ss2}$
by (*simp* *only*: *ctxt-pt.supp-def terms-pt.supp-def*) *auto*

lemma *supp-More*:

$\text{ctxt-pt.supp } (\text{More } f \text{ ss1 } C \text{ ss2}) =$
 $(\bigcup s \in \text{set ss1}. \text{term-pt.supp } s) \cup \text{ctxt-pt.supp } C \cup (\bigcup s \in \text{set ss2}. \text{term-pt.supp } s)$
by (*simp* *add*: *supp-More'*)

lemma *supp-vars-ctxt-eq*:
fixes $C :: ('f, 'v :: \text{infinite}) \text{ ctxt}$
shows $\text{ctxt-pt.supp } C = \text{vars-ctxt } C$
by (*induct* C) (*auto simp: supp-Hole supp-More supp-vars-term-eq*)

lemma *term-apply-ctxt-eqv* [*eqvt*]:
fixes $p :: ('v :: \text{infinite}) \text{ perm}$
shows $p \cdot (C\langle t \rangle) = (p \cdot C)\langle p \cdot t \rangle$
by (*induct* C) *simp+*

interpretation *rule-pt*: *prod-pt permute-term permute-term ..*
interpretation *rules-pt*: *list-pt rule-pt.permute-prod ..*

adhoc-overloading
PERMUTE *rule-pt.permute-prod rules-pt.permute-list* **and**
FRESH *rule-pt.fresh rule-pt.fresh-set rules-pt.fresh rules-pt.fresh-set*

lemma *supp-vars-rule-eq*:
fixes $r :: ('f, 'v :: \text{infinite}) \text{ rule}$
shows $\text{rule-pt.supp } r = \text{vars-term } (\text{fst } r) \cup \text{vars-term } (\text{snd } r)$
by (*cases* r) (*simp add: supp-vars-term-eq*)

interpretation *trs-pt*: *rel-pt permute-term ..*

adhoc-overloading
PERMUTE *trs-pt.permute-set* **and**
FRESH *trs-pt.fresh*

interpretation *term-set-pt*: *set-pt permute-term ..*

adhoc-overloading
PERMUTE *term-set-pt.permute-set* **and**
FRESH *term-set-pt.fresh*

lemma *rule-mem-trs-iff* [*iff*]:
fixes $p :: ('v :: \text{infinite}) \text{ perm}$ **and** $R :: ('f, 'v) \text{ term rel}$
shows $(p \cdot s, p \cdot t) \in p \cdot R \longleftrightarrow (s, t) \in R$
unfolding *rule-pt.permute-prod.simps* [*symmetric*] *trs-pt.mem-permute-iff* ..

lemma *inv-rule-mem-trs-simps* [*simp*]:
fixes $p :: ('v :: \text{infinite}) \text{ perm}$ **and** $R :: ('f, 'v) \text{ term rel}$
shows $(-p \cdot s, -p \cdot t) \in R \longleftrightarrow (s, t) \in p \cdot R$
and $(s, t) \in -p \cdot R \longleftrightarrow (p \cdot s, p \cdot t) \in R$
unfolding *rule-pt.permute-prod.simps* [*symmetric*]
by (*metis* *trs-pt.inv-mem-simps*(1))
(metis *trs-pt.inv-mem-simps*(2))

lemma *symcl-trs-eqv* [*eqvt*]: $\pi \cdot R^{\leftrightarrow} = (\pi \cdot R)^{\leftrightarrow}$

```

by (auto simp: eqvt)
(meson converse-iff inv-rule-mem-trs-simps(1))+

lemma term-apply-subst-Var-Rep-perm [simp]:
   $t \cdot \text{sop } p = p \cdot t$ 
by (induct t) (simp add: permute-atom-def)+

lemma rstep-permute-subset:
   $\text{rstep } (p \cdot R) \subseteq \text{rstep } R$ 
proof (rule subrelI)
  fix s t
  assume  $(s, t) \in \text{rstep } (p \cdot R)$ 
  then show  $(s, t) \in \text{rstep } R$ 
  proof (rule rstepE)
    fix l r C  $\sigma$ 
    presume  $(l, r) \in p \cdot R$  and [simp]:  $s = C\langle l \cdot \sigma \rangle$   $t = C\langle r \cdot \sigma \rangle$ 
    then have  $(-p \cdot l, -p \cdot r) \in R$  by simp
    then have  $((-p \cdot l) \cdot \text{sop } p, (-p \cdot r) \cdot \text{sop } p) \in \text{rstep } R$  by blast
    then show  $(s, t) \in \text{rstep } R$  by auto
  qed auto
qed

The rewrite relation is invariant under variable renamings of the given TRS.

lemma rstep-permute [simp]:
   $\text{rstep } (p \cdot R) = \text{rstep } R$ 
proof
  show  $\text{rstep } (p \cdot R) \subseteq \text{rstep } R$  by (rule rstep-permute-subset)
next
  have  $\text{rstep } R = \text{rstep } (-p \cdot p \cdot R)$  by simp
  also have  $\dots \subseteq \text{rstep } (p \cdot R)$  by (rule rstep-permute-subset)
  finally show  $\text{rstep } R \subseteq \text{rstep } (p \cdot R)$  .
qed

lemma permute-rstep-subset:
   $p \cdot \text{rstep } R \subseteq \text{rstep } R$ 
proof (rule subrelI)
  fix s t
  assume  $(s, t) \in p \cdot \text{rstep } R$ 
  then have  $(-p \cdot s, -p \cdot t) \in \text{rstep } R$  by simp
  then show  $(s, t) \in \text{rstep } R$ 
  proof (induct  $x \equiv -p \cdot s$   $y \equiv -p \cdot t$ )
    case (IH C  $\sigma$  l r)
    then have  $(p \cdot l, p \cdot r) \in p \cdot R$ 
    and [simp]:  $s = p \cdot C\langle l \cdot \sigma \rangle$   $t = p \cdot C\langle r \cdot \sigma \rangle$  by simp+
    then have  $((p \cdot C)\langle (p \cdot l) \cdot (\text{conjugate-subst } p \sigma) \rangle,$ 
       $(p \cdot C)\langle (p \cdot r) \cdot (\text{conjugate-subst } p \sigma) \rangle) \in \text{rstep } (p \cdot R)$  by blast
    then show  $(s, t) \in \text{rstep } R$  by (simp add: eqvt)
  qed
qed

```

```

lemma permute-rstep [simp]:
   $p \cdot \text{rstep } R = \text{rstep } R$ 
proof
  show  $p \cdot \text{rstep } R \subseteq \text{rstep } R$  by (rule permute-rstep-subset)
next
  have  $\text{rstep } R = p \cdot -p \cdot \text{rstep } R$  by simp
  also have  $\dots \subseteq p \cdot \text{rstep } R$ 
    using permute-rstep-subset [of  $-p \ R$ ] by auto
  finally show  $\text{rstep } R \subseteq p \cdot \text{rstep } R$  .
qed

lemma rstep-eqv [eqvt]:
   $p \cdot \text{rstep } R = \text{rstep } (p \cdot R)$ 
by simp

lemma rstep-imp-perm-rstep:
   $(s, t) \in \text{rstep } R \implies (p \cdot s, p \cdot t) \in \text{rstep } R$ 
by (subst (asm) rule-mem-trs-iff [symmetric]) (simp add: eqvt)

lemma perm-rstep-imp-rstep:
   $(p \cdot s, p \cdot t) \in \text{rstep } R \implies (s, t) \in \text{rstep } R$ 
by (subst (asm) rule-mem-trs-iff [symmetric, of - - -p]) (simp add: eqvt)

lemma rstep-permute-iff [iff]:
   $(p \cdot s, p \cdot t) \in \text{rstep } R \longleftrightarrow (s, t) \in \text{rstep } R$ 
by (metis perm-rstep-imp-rstep rstep-imp-perm-rstep)

lemma term-apply-subst-Var-Abs-perm:
   $f \in \text{perms} \implies t \cdot (\text{Var} \circ f) = \text{Abs-perm } f \cdot t$ 
by (metis Abs-perm-inverse term-apply-subst-Var-Rep-perm)

lemma finite-term-supp: finite (term-pt.supp t)
unfolding supp-vars-term-eq
by (induct t) simp+
```

lemma *finite-rule-supp*: *finite* (*rule-pt.supp* (*l*, *r*))
by (*simp add: finite-term-supp*)

interpretation *term-fs*: *finitely-supported permute-term*
by *standard* (*rule finite-term-supp*)

interpretation *rule-fs*: *finitely-supported rule-pt.permute-prod*
by *standard* (*auto simp: finite-rule-supp finite-term-supp*)

lemma *vars-rule-disjoint*:
fixes $l \ r \ u \ v :: ('f, 'v :: \text{infinite}) \text{ term}$
shows $\exists p. \text{vars-rule } (p \cdot (l, r)) \cap \text{vars-rule } (u, v) = \{\}$
proof –

```

from rule-fs.suppl-fresh-set obtain p
  where rule-pt.suppl (p • (l, r)) # (u, v) by blast
from rule-pt.fresh-set-disjoint [OF this]
  show ?thesis
  by (auto simp: suppl-vars-rule-eq vars-rule-def)
qed

lemma vars-term-eqv [eqvt]:
   $\pi \cdot \text{vars-term } t = \text{vars-term } (\pi \cdot t)$ 
  by (simp add: suppl-vars-term-eq [symmetric] eqvt)

lemma permute-term-subst-apply-term:
   $(\pi \cdot t) \cdot \sigma = t \cdot (\sigma \circ \text{Rep-perm } \pi)$ 
  by (induct t) (simp-all add: permute-atom-def)

lemma permute-subst-subst-compose:
   $\pi \cdot \sigma = \sigma \circ_s \text{sop } \pi$ 
  by (rule ext) (simp add: subst-compose-def permute-subst-def)

lemma vars-rule-eqv [eqvt]:
   $\pi \cdot \text{vars-rule } r = \text{vars-rule } (\pi \cdot r)$ 
  by (simp add: vars-rule-def) (metis rule-fs.suppl-eqv suppl-vars-rule-eq)

lemma fun-poss-perm-simp [simp]:
   $\text{fun-poss } (\pi \cdot t) = \text{fun-poss } t$ 
  by (induct t) auto

lemma poss-perm-simp [simp]:
   $\text{poss } (\pi \cdot t) = \text{poss } t$ 
  by (induct t) auto

definition permute-pos :: 'a perm  $\Rightarrow$  pos  $\Rightarrow$  pos
where
  permute-pos  $\pi$  (p :: pos) = p

interpretation pos-pure: pure permute-pos
  by standard (simp-all add: permute-pos-def)

adhoc-overloading
  PERMUTE permute-pos

interpretation pos-set-pt: set-pt permute-pos ..

adhoc-overloading
  PERMUTE pos-set-pt.permute-set

interpretation pos-set-pure: pure pos-set-pt.permute-set
  by standard (simp add: permute-pos-def pos-set-pt.permute-set-def)

```

lemma *fun-poss-eqvt* [eqvt]:
 $\pi \cdot \text{fun-poss } t = \text{fun-poss } (\pi \cdot t)$
by *simp*

lemma *poss-eqvt* [eqvt]:
 $\pi \cdot \text{poss } t = \text{poss } (\pi \cdot t)$
by *simp*

lemma *subt-at-eqvt* [eqvt]:
 $p \in \text{poss } t \implies \pi \cdot (t \mid p) = (\pi \cdot t) \mid p$
by (*induct t p rule: subt-at.induct*)
(auto simp del: subt-at-Cons-distr)

lemma *subst-variants-id*:
assumes *variants*: $\sigma \circ_s \sigma' = \tau \tau \circ_s \tau' = \sigma$
and $x: x \in \bigcup (\text{vars-term } \sigma \text{ ' } V)$ (**is** $x \in ?D$)
shows $(\text{the-Var} \circ \tau') ((\text{the-Var} \circ \sigma') x) = x$ (**is** $? \tau' (? \sigma' x) = x$)
proof –
note $* [\text{simp}] = \text{subst-variants-imp-eq} [OF \text{ variants}]$
{ **fix** y
assume $**$: $x \in \text{vars-term } (\sigma y)$
then have $\text{var}: \text{is-Var } (\sigma' x)$
using *variants-imp-is-Var* [OF $*$] **by** *simp*
have $\bigwedge \mu. \mu x = \text{Var } x \vee \sigma y \cdot \mu \neq \sigma y$
using $**$ **by** (*metis subst-apply-term-empty term-subst-eq-rev*)
then have $\bigwedge \mu \nu. (\nu x) \cdot \mu = \text{Var } x \vee \sigma y \cdot \nu \cdot \mu \neq \sigma y$
by (*metis eval-term.simps(1) subst-subst*)
then have $? \tau' (? \sigma' x) = x$
using var **and** $*$ **by** (*metis term.collapse(1) eval-term.simps(1) term.sel(1)*)
o-def) **}**
then show $?thesis$ **using** x **by** *auto*
qed

lemma *subst-variants-image-subset*:
assumes *variants*: $\sigma \circ_s \sigma' = \tau \tau \circ_s \tau' = \sigma$
shows $(\text{the-Var} \circ \sigma') \text{ ' } \bigcup (\text{vars-term } \sigma \text{ ' } V) \subseteq \bigcup (\text{vars-term } \tau \text{ ' } V)$
(is $? \sigma' \text{ ' } ?S \subseteq ?T$)
proof
note $* = \text{subst-variants-imp-eq} [OF \text{ variants}]$
have *subst-fun-range-subset*: $?S \subseteq \text{subst-fun-range } \sigma$ **by** (*auto simp: subst-fun-range-def*)
note *is-Var* = *subst-variants-imp-is-Var* [OF *variants*]

fix x_0
assume $x_0 \in ? \sigma' \text{ ' } ?S$
then obtain x **where** $**$: $? \sigma' x = x_0$ **and** $x \in ?S$ **by** *blast*
then obtain x' **where** $x \in \text{vars-term } (\sigma x')$ **and** $x' \in V$ **by** (*auto*)
then have $\sigma x' \supseteq \text{Var } x$ **by** (*metis supteq-Var*)
then have $\sigma x' \cdot \sigma' \supseteq \sigma' x$ **by** *auto*
then have $\tau x' \supseteq \sigma' x$ **by** (*simp add: **)

moreover
from *is-Var* **and** *subst-fun-range-subset* **and** $\langle x \in ?S \rangle$ **have** *is-Var* $(\sigma' x)$ **by**
auto
ultimately have $? \sigma' x \in \text{vars-term } (\tau x')$
by (*metis term.collapse(1) comp-def subteq-Var-imp-in-vars-term*)
then have $? \sigma' x \in ?T$ **using** $\langle x' \in V \rangle$ **by** (*auto simp:*)
then show $x_0 \in ?T$ **unfolding** **** .**
qed

lemma *subst-variants-imp-image-eq*:
assumes $\sigma \circ_s \sigma' = \tau$ **and** $\tau \circ_s \tau' = \sigma$
shows $(\text{the-Var} \circ \sigma') ' \bigcup (\text{vars-term } ' \sigma ' V) = \bigcup (\text{vars-term } ' \tau ' V)$
(is $? \sigma' ' ?S = ?T$ **)**
proof
show $? \sigma' ' ?S \subseteq ?T$
using *subst-variants-image-subset [OF assms, of V]* **by** (*simp*)
next
let $? \tau' = \text{the-Var} \circ \tau'$
have $? \sigma' ' ? \tau' ' ?T \subseteq ? \sigma' ' ?S$
using *subst-variants-image-subset [OF assms(2, 1), of V]* **by** (*metis image-mono*)
moreover have $? \sigma' ' ? \tau' ' ?T = ?T$
using *subst-variants-id [OF assms(2, 1), of - V]*
by (*auto simp: o-def*) (*metis, metis (full-types) UN-I image-eqI*)
ultimately show $?T \subseteq ? \sigma' ' ?S$ **by** *simp*
qed

If two variables are substitution instances of each other, then they only differ by a variable renaming.

lemma *subst-variants-imp-perm*:
fixes $\sigma \tau :: ('f, 'v :: \text{infinite}) \text{subst}$
assumes *variants*: $\sigma \circ_s \sigma' = \tau$ $\tau \circ_s \tau' = \sigma$
and *finite*: *finite* (*subst-domain* σ) *finite* (*subst-domain* τ)
shows $\exists \pi. \pi \cdot \sigma = \tau$
proof –
define D **where** $D = \text{subst-domain } \sigma \cup \text{subst-domain } \tau$
define D_σ **where** $D_\sigma = \bigcup (\text{vars-term } ' \sigma ' D)$
define D_τ **where** $D_\tau = \bigcup (\text{vars-term } ' \tau ' D)$
let $? \sigma' = \text{the-Var} \circ \sigma'$
let $? \tau' = \text{the-Var} \circ \tau'$

note $*$ = *subst-variants-imp-eq [OF variants]*
note *is-Var* = *subst-variants-imp-is-Var [OF variants]*

have *finite*: *finite* D_σ **using** *finite* **by** (*auto simp: D_σ-def D-def*)
have *subst-fun-range-subset*: $D_\sigma \subseteq \text{subst-fun-range } \sigma$ **by** (*auto simp: D_σ-def subst-fun-range-def*)
have *id*: $\bigwedge x. x \notin D \implies \sigma' x = \text{Var } x$ $\bigwedge x. x \notin D \implies \tau' x = \text{Var } x$
 $\bigwedge x. x \notin D \implies \sigma x = \text{Var } x$ $\bigwedge x. x \notin D \implies \tau x = \text{Var } x$

```

using * by (auto simp: D-def subst-domain-def) (metis eval-term.simps(1))+

have ?σ' ' Dσ = Dτ
  using subst-variants-imp-image-eq [OF variants] by (simp add: Dσ-def Dτ-def)
moreover have inj-on ?σ' Dσ
  by (rule inj-on-inverseI [of - ?τ]) (insert subst-variants-id [OF variants], simp
add: Dσ-def)
ultimately have bij-betw ?σ' Dσ Dτ by (auto simp: bij-betw-def)
from bij-betw-extend [OF this, of UNIV] and finite obtain g
  where finite {x. g x ≠ x} and g-id: (∀ x ∈ UNIV - (Dσ ∪ Dτ). g x = x)
  and g-eq: ∀ x ∈ Dσ. g x = (the-Var ∘ σ') x and bij g by blast
then have perm: g ∈ perms by (auto simp: perms-def)
have Abs-perm g · σ = τ
proof (rule ext)
  fix x
  have Abs-perm g · σ x = τ x
proof (cases x ∈ D)
  assume x ∈ D
  then have vars-term (σ x) ⊆ Dσ by (auto simp: Dσ-def)
  with g-eq and term-subst-eq [of σ x Var ∘ g σ']
  have σ x · (Var ∘ g) = σ x · σ'
  by auto (metis subst-fun-range-subset(1) term.collapse(1) is-Var(1) set-rev-mp)
  with perm and * show ?thesis by (simp add: term-apply-subst-Var-Abs-perm)
next
  have Dτ - D ⊆ Dσ - D
  proof -
    have id: ?τ' ' (Dτ - D) = Dτ - D
    using id by auto
    have ?τ' ' (Dτ - D) ⊆ ?τ' ' Dτ by blast
    also have ... ⊆ Dσ
    using subst-variants-image-subset [OF variants(2, 1), of D] by (simp add:
Dτ-def Dσ-def)
    finally show ?thesis unfolding id by auto
  qed
  then have **: Dτ - Dσ ⊆ D by blast
  assume x ∉ D
  moreover then have Abs-perm g · x = x
    using g-eq and id and g-id and perm and **
    by (cases x ∈ Dσ) (auto simp: permute-atom-def Abs-perm-inverse)
  ultimately show ?thesis by (simp add: id)
qed
then show (Abs-perm g · σ) x = τ x by (simp add: permute-subst-def)
qed
then show ?thesis ..
qed

lemma is-mgu-imp-perm:
  fixes E :: ('f, 'v :: infinite) equations
  assumes mgu: is-mgu σ E is-mgu τ E

```

```

    and finite: finite (subst-domain  $\sigma$ ) finite (subst-domain  $\tau$ )
  shows  $\exists \pi. \pi \cdot \sigma = \tau$ 
proof -
  obtain  $\sigma' \tau' :: ('f, 'v)$  subst
  where  $\sigma \circ_s \sigma' = \tau$  and  $\tau \circ_s \tau' = \sigma$ 
  using mgu by (auto simp: is-mgu-def unifiers-def) metis
  from subst-variants-imp-perm [OF this finite]
  show ?thesis .
qed

```

```

lemma unifiers-perm-simp [simp]:
   $\pi \cdot \sigma \in \text{unifiers } E \longleftrightarrow \sigma \in \text{unifiers } E$ 
  by (auto simp: unifiers-def permute-subst-subst-compose)

```

```

lemma inv-Rep-perm-simp [simp]:
  fixes  $x :: 'v :: \text{infinite}$ 
  shows  $-\pi \cdot \text{Rep-perm } \pi x = x$ 
  by (simp add: permute-atom-def Rep-perm-uminus)
  (metis Rep-perm-uminus atom-pt.permute-minus-cancel(2) permute-atom-def)

```

```

lemma permute-subst-conjugate-subst [simp]:
   $(\pi \cdot \sigma) \circ_s \text{conjugate-subst } \pi \tau = \sigma \circ_s (\tau \circ_s (\text{Var} \circ \text{Rep-perm } \pi))$ 
  apply (rule ext)
  apply (simp add: subst-conjugate-pt.permute-fun-def)
  apply (simp add: permute-subst-subst-compose ac-simps)
  apply (simp add: subst-compose-def)
done

```

Every renaming of an mgu is again an mgu.

```

lemma is-mgu-perm:
  fixes  $E :: ('f, 'v :: \text{infinite})$  equations
  assumes is-mgu  $\sigma E$ 
  shows is-mgu  $(\pi \cdot \sigma) E$ 
proof (unfold is-mgu-def, intro conjI ballI)
  show  $\pi \cdot \sigma \in \text{unifiers } E$  using assms by (simp add: is-mgu-def)
next
  fix  $\tau :: ('f, 'v)$  subst
  assume  $\tau \in \text{unifiers } E$ 
  then obtain  $\gamma :: ('f, 'v)$  subst
  where  $\tau = \sigma \circ_s \gamma$  using assms by (auto simp: is-mgu-def)
  then have  $\tau \circ_s \text{sop } \pi = (\pi \cdot \sigma) \circ_s \text{conjugate-subst } \pi \gamma$ 
  by (simp add: ac-simps)
  then have  $\tau \circ_s \text{sop } \pi \circ_s \text{sop } (-\pi) =$ 
     $(\pi \cdot \sigma) \circ_s \text{conjugate-subst } \pi \gamma \circ_s \text{sop } (-\pi)$  by simp
  then have  $\tau = (\pi \cdot \sigma) \circ_s \text{conjugate-subst } \pi \gamma \circ_s \text{sop } (-\pi)$ 
  by (simp add: subst-compose-def)
  then have  $\tau = (\pi \cdot \sigma) \circ_s (\text{conjugate-subst } \pi \gamma \circ_s \text{sop } (-\pi))$ 
  unfolding subst-compose-assoc .
  then show  $\exists \gamma :: ('f, 'v)$  subst.  $\tau = (\pi \cdot \sigma) \circ_s \gamma$  by blast

```

qed

lemma *Rep-perm-image*:

Rep-perm $\pi \cdot A = \pi \cdot A$

by (*metis atom-set-pt.permute-set-eq-image image-cong permute-atom-def*)

lemma *vars-term-perm-eq*:

assumes $\forall x \in \text{vars-term } t. \pi \cdot x = \pi' \cdot x$

shows $\pi \cdot t = \pi' \cdot t$

using *assms* **by** (*induct t*) *simp-all*

lemma *vars-rule-perm-eq*:

assumes $\forall x \in \text{vars-rule } r. \pi \cdot x = \pi' \cdot x$

shows $\pi \cdot r = \pi' \cdot r$

using *assms* **by** (*cases r*) (*auto simp: vars-rule-def vars-term-perm-eq eqvt*)

lemma *rule-variants-imp-perm*:

assumes *disj*: $\text{vars-rule } (\pi_1 \cdot r) \cap \text{vars-rule } (\pi_2 \cdot r') = \{\}$ (**is** $?V \cap ?V' = \{\}$)

$\text{vars-rule } (\pi_3 \cdot r) \cap \text{vars-rule } (\pi_4 \cdot r') = \{\}$ (**is** $?W \cap ?W' = \{\}$)

shows $\exists \pi. \pi \cdot \pi_3 \cdot r = \pi_1 \cdot r \wedge \pi \cdot \pi_4 \cdot r' = \pi_2 \cdot r'$

proof –

let $?f = \text{Rep-perm } (\pi_1 + -\pi_3)$

let $?g = \text{Rep-perm } (\pi_2 + -\pi_4)$

have *bij* $?f$ **by** (*metis bij-Rep-perm*)

then have *bij*: *bij-betw* $?f ?W$ ($?f \cdot ?W$) **by** (*auto elim: bij-betw-subset*)

have *bij* $?g$ **by** (*metis bij-Rep-perm*)

then have *bij'*: *bij-betw* $?g ?W'$ ($?g \cdot ?W'$) **by** (*auto elim: bij-betw-subset*)

define *f* **where** $f \equiv \lambda x. \text{if } x \in \text{vars-rule } (\pi_3 \cdot r) \text{ then } ?f x \text{ else } x$

define *g* **where** $g \equiv \lambda x. \text{if } x \in \text{vars-rule } (\pi_4 \cdot r') \text{ then } ?g x \text{ else } x$

define *h* **where** $h \equiv \lambda x. \text{if } x \in \text{vars-rule } (\pi_3 \cdot r) \text{ then } f x \text{ else if } x \in \text{vars-rule } (\pi_4 \cdot r') \text{ then } g x \text{ else } x$

have *bij*: *bij-betw* $f ?W$ ($f \cdot ?W$) **using** *bij* **by** (*auto simp: f-def bij-betw-def inj-on-def*)

have *bij'*: *bij-betw* $g ?W'$ ($g \cdot ?W'$) **using** *bij'* **by** (*auto simp: g-def bij-betw-def inj-on-def*)

have *: $f \cdot ?W' = ?W' g \cdot ?W = ?W$ **using** *disj*(2) **by** (*auto simp: f-def g-def*)

have **: $h \cdot ?W' = g \cdot ?W' h \cdot ?W = f \cdot ?W$ **using** *disj*(2) **by** (*auto simp: h-def*)

have ***: $f \cdot ?W = ?V g \cdot ?W' = ?V'$ **using** *disj* **by** (*auto simp: f-def g-def eqvt Rep-perm-image*)

have *bij-betw* $h ?W ?V$

using *bij* **by** (*auto simp: *** bij-betw-def h-def inj-on-def*)

moreover have *bij-betw* $h ?W' ?V'$

using *bij'* **and** *disj*

apply (*simp add: * ** *** bij-betw-def h-def inj-on-def*)

apply (*intro conjI set-eqI iffI*)

subgoal by *auto*

subgoal by *auto* (*metis ***(2) imageI*)

```

    subgoal by auto (metis ***(2) Compl-eq Diff-eq Diff-triv Int-commute)
  done
  moreover have  $h \cdot ?W \cap h \cdot ?W' = \{\}$ 
    unfolding ** *** using disj(1) .
  ultimately have bij-betw  $h$   $(?W \cup ?W')$   $(?V \cup ?V')$ 
    using bij-betw-combine [of  $h$   $?W$   $?V$   $?W'$   $?V'$ ]
    unfolding ** *** by blast
  from conjI [THEN bij-betw-extend [OF this, of UNIV, simplified], OF finite-vars-rule
finite-vars-rule]
    obtain  $b$ 
      where finite  $\{x. b \ x \neq x\}$ 
      and neg:  $\forall x \in \text{UNIV} - ((?W \cup ?W') \cup (?V \cup ?V')). b \ x = x$ 
      and eq:  $\forall x \in ?W \cup ?W'. b \ x = h \ x$ 
      and bij  $b$ 
      by auto
  then have perm:  $b \in \text{perms}$  by (auto simp: perms-def)
  have Abs-perm  $b \cdot \pi_3 \cdot r = \pi_1 \cdot r$ 
  proof -
    have Abs-perm  $b \cdot \pi_3 \cdot r = \text{Abs-perm } ?f \cdot \pi_3 \cdot r$ 
    apply (rule vars-rule-perm-eq)
    apply (insert eq perm)
    apply (simp add: h-def f-def g-def Abs-perm-inverse Rep-perm permute-atom-def
      split: if-splits)
    done
    also have  $\dots = \pi_1 \cdot r$  by (simp add: Rep-perm-inverse)
    finally show ?thesis .
  qed
  moreover have Abs-perm  $b \cdot \pi_4 \cdot r' = \pi_2 \cdot r'$ 
  proof -
    have Abs-perm  $b \cdot \pi_4 \cdot r' = \text{Abs-perm } ?g \cdot \pi_4 \cdot r'$ 
    apply (rule vars-rule-perm-eq)
    apply (insert eq perm disj)
    apply (auto simp add: h-def f-def g-def Abs-perm-inverse Rep-perm permute-atom-def
      split: if-splits)
    done
    also have  $\dots = \pi_2 \cdot r'$  by (simp add: Rep-perm-inverse)
    finally show ?thesis .
  qed
  ultimately show ?thesis by blast
qed

lemma poss-perm-prod-simps [simp]:
  poss (fst  $(\pi \cdot r)$ ) = poss (fst  $r$ )
  poss (snd  $(\pi \cdot r)$ ) = poss (snd  $r$ )
  by (cases  $r$ , auto simp: eqvt) +

lemma ctxt-of-pos-term-eqvt [eqvt]:
  assumes  $p \in \text{poss } t$ 
  shows  $\pi \cdot (\text{ctxt-of-pos-term } p \ t) = \text{ctxt-of-pos-term } p \ (\pi \cdot t)$ 

```

```

using assms by (induct t arbitrary: p) (auto simp: take-map drop-map)

lemma finite-subst-domain-sop:
  finite (subst-domain (sop  $\pi$ ))
by (auto simp: subst-domain-def finite-Rep-perm)

lemma fun-poss-perm-iff [simp]:
   $p \in \text{fun-poss } (\pi \cdot t) \longleftrightarrow p \in \text{fun-poss } t$ 
by (induct t) auto

lemma perm-rstep-conv [simp]:
   $\pi \cdot p \in \text{rstep } R \longleftrightarrow p \in \text{rstep } R$ 
by (metis rstep-eqvt rstep-permute trs-pt.mem-permute-iff)

lemma perm-rstep-perm:
  assumes  $(\pi \cdot s, t) \in \text{rstep } R$ 
  shows  $\exists u. t = \pi \cdot u$ 
using assms by (metis term-pt.permute-minus-cancel(1))

lemma perm-rsteps-perm:
  assumes  $(\pi \cdot s, t) \in (\text{rstep } R)^*$ 
  shows  $\exists u. t = \pi \cdot u$ 
using assms by (metis term-pt.permute-minus-cancel(1))

lemma perm-rsteps-conv [simp]:
   $\pi \cdot p \in (\text{rstep } R)^* \longleftrightarrow p \in (\text{rstep } R)^*$ 
by (metis perm-rstep-conv rstep-rtrancl-idemp)

lemma perm-join-conv [simp]:
   $\pi \cdot p \in (\text{rstep } R)^\downarrow \longleftrightarrow p \in (\text{rstep } R)^\downarrow$  (is  $?L = ?R$ )
proof
  assume  $?L$ 
  then obtain u
    where  $(fst (\pi \cdot p), u) \in (\text{rstep } R)^*$  and  $(snd (\pi \cdot p), u) \in (\text{rstep } R)^*$ 
    by (metis joinE surjective-pairing)
  moreover then obtain v
    where [simp]:  $u = \pi \cdot v$  by (auto dest: perm-rsteps-perm simp: eqvt [symmetric])
  ultimately have  $(fst p, v) \in (\text{rstep } R)^*$  and  $(snd p, v) \in (\text{rstep } R)^*$ 
    by (simp-all add: eqvt [symmetric])
  then show  $?R$  by (metis joinI surjective-pairing)
next
  assume  $?R$ 
  then obtain u
    where  $(fst p, u) \in (\text{rstep } R)^*$  and  $(snd p, u) \in (\text{rstep } R)^*$ 
    by (metis joinE surjective-pairing)
  then have  $\pi \cdot (fst p, u) \in (\text{rstep } R)^*$  and  $\pi \cdot (snd p, u) \in (\text{rstep } R)^*$  by auto
  then show  $?L$  by (metis joinI rule-pt.permute-prod-eqvt surjective-pairing)
qed

```

lemma *permuted-rule-in-variants*:
assumes $(p \cdot s, p \cdot t) \in R$
shows $(s, t) \in \text{rule-pt.variants}$ “ R
using *assms*
by (*auto simp: rule-pt.variants-def*)
(metis rule-pt.permute-minus-cancel(2) rule-pt.permute-prod.simps)

lemma *in-fun-poss-eqvt*:
assumes $p \in \text{fun-poss } t$
shows $p \in \text{fun-poss } (\pi \cdot t)$
using *assms* **by** (*induct t arbitrary: p*) *auto*

Lists have an infinite universe.

instance *list* :: (*type*) *infinite* **by** *standard (rule infinite-UNIV-listI)*

Two terms are variants iff they are substitution instances of each other.

lemma *term-variants-iff*:
fixes $s \ t :: ('f, 'v :: \text{infinite}) \text{ term}$
shows $(\exists \pi. \pi \cdot s = t) \longleftrightarrow (\exists (\sigma :: ('f, 'v) \text{ subst}) (\tau :: ('f, 'v) \text{ subst}). s \cdot \sigma = t$
 $\wedge t \cdot \tau = s)$
(is ?L = ?R)

proof

assume $?L$
then obtain π **where** $*: \pi \cdot s = t$..
have $s \cdot (\text{Var} \circ \text{Rep-perm } \pi) = t$ **by** (*simp add: **)
moreover have $t \cdot (\text{Var} \circ \text{Rep-perm } (-\pi)) = s$ **by** (*simp add: * [symmetric]*)
ultimately show $?R$ **by** *blast*

next

assume $?R$
then obtain $\sigma \ \tau :: ('f, 'v) \text{ subst}$
where *variants: $s \cdot \sigma = t \ t \cdot \tau = s$* **by** *blast*
from *variants-imp-bij-betw-vars [OF variants]*
have *bij-betw (the-Var \circ σ) (vars-term s) (vars-term t) .*
from *bij-betw-extend [OF this, of UNIV]* **obtain** $g :: 'v \Rightarrow 'v$
where *finite $\{x. g \ x \neq x\}$* **and** $*: \forall x \in \text{vars-term } s. g \ x = (\text{the-Var} \circ \sigma) \ x$ **and**
bij g **by** *auto*
then have $g \in \text{perms}$ **by** (*simp add: perms-def*)
then have $*: s \cdot (\text{Var} \circ g) = \text{Abs-perm } g \cdot s$ **by** (*rule term-apply-subst-Var-Abs-perm*)
from $*$ **have** $\forall x \in \text{vars-term } s. (\text{Var} \circ g) \ x = \sigma \ x$
using *variants-imp-is-Var [OF variants]* **by** *simp*
from *term-subst-eq-conv [THEN iffD2, OF this]*
show $?L$ **unfolding** *variants* **and** $**$..

qed

lemma *rstep-pos-permute [simp]*:
 $\text{rstep-pos } (\pi \cdot R) \ p = \text{rstep-pos } R \ p$
proof
show $\text{rstep-pos } (\pi \cdot R) \ p \subseteq \text{rstep-pos } R \ p$
proof (*rule subrelI*)

```

fix  $s\ t$ 
assume  $(s, t) \in \text{rstep-pos } (\pi \cdot R)\ p$ 
then show  $(s, t) \in \text{rstep-pos } R\ p$ 
proof (cases)
  fix  $l\ r\ \sigma$ 
  assume  $(l, r) \in \pi \cdot R$ 
    and  $[simp]: s \mid -\ p = l \cdot \sigma\ t = (\text{ctxt-of-pos-term } p\ s)\langle r \cdot \sigma \rangle$ 
    and  $p: p \in \text{poss } s$ 
  then have  $(-\pi \cdot l, -\pi \cdot r) \in -\pi \cdot \pi \cdot R$  by simp
  then have  $(-\pi \cdot l, -\pi \cdot r) \in R$  by simp
  from rstep-pos.intros [OF this p, of sop  $\pi \circ_s \sigma$ ]
  show  $(s, t) \in \text{rstep-pos } R\ p$  by simp
qed
qed
next
show  $\text{rstep-pos } R\ p \subseteq \text{rstep-pos } (\pi \cdot R)\ p$ 
proof (rule subrelI)
  fix  $s\ t$ 
  assume  $(s, t) \in \text{rstep-pos } R\ p$ 
  then show  $(s, t) \in \text{rstep-pos } (\pi \cdot R)\ p$ 
  proof (cases)
    fix  $l\ r\ \sigma$ 
    assume  $(l, r) \in R$  and  $[simp]: s \mid -\ p = l \cdot \sigma\ t = (\text{ctxt-of-pos-term } p\ s)\langle r \cdot \sigma \rangle$ 
    and  $p: p \in \text{poss } s$ 
    then have  $(\pi \cdot l, \pi \cdot r) \in \pi \cdot R$  by simp
    from rstep-pos.intros [OF this p, of sop  $(-\pi) \circ_s \sigma$ ]
    show  $(s, t) \in \text{rstep-pos } (\pi \cdot R)\ p$  by simp
  qed
qed
qed

lemma rstep-pos-subset-permute-rstep-pos:
   $\text{rstep-pos } R\ p \subseteq \pi \cdot \text{rstep-pos } R\ p$ 
proof (rule subrelI)
  fix  $s\ t$ 
  assume  $(s, t) \in \text{rstep-pos } R\ p$ 
  then have  $(s, t) \in \text{rstep-pos } (\pi \cdot R)\ p$  by auto
  then show  $(s, t) \in \pi \cdot \text{rstep-pos } R\ p$ 
  proof (cases)
    case (rule l r  $\sigma$ )
    then have  $p: p \in \text{poss } (-\pi \cdot s)$ 
    and  $*: -\pi \cdot s \mid -\ p = -\pi \cdot l \cdot \text{sop } \pi \circ_s \sigma \circ_s \text{sop } (-\pi)$  by (auto simp: eqvt
    [symmetric])
    have  $(l, r) \in \pi \cdot R$  by fact
    then have  $(\pi \cdot (-\pi) \cdot l, \pi \cdot (-\pi) \cdot r) \in \pi \cdot R$  by simp+
    then have  $(-\pi \cdot l, -\pi \cdot r) \in R$  by simp
    from rstep-pos.intros [OF this p *] and  $p$ 
    show ?thesis unfolding rule by (auto simp: eqvt [symmetric])
  qed

```

qed

lemma *rstep-pos-eqv* [eqvt]:

$\pi \cdot \text{rstep-pos } R \ p = \text{rstep-pos } (\pi \cdot R) \ p$

proof

show $\text{rstep-pos } (\pi \cdot R) \ p \subseteq \pi \cdot \text{rstep-pos } R \ p$

using *rstep-pos-subset-permute-rstep-pos* **by** *simp*

next

from *rstep-pos-subset-permute-rstep-pos* [of $R \ p \ -\pi$]

have $\pi \cdot \text{rstep-pos } R \ p \subseteq \pi \cdot -\pi \cdot \text{rstep-pos } R \ p$ **by** (*metis trs-pt.permute-set-subset*)

then show $\pi \cdot \text{rstep-pos } R \ p \subseteq \text{rstep-pos } (\pi \cdot R) \ p$ **by** *simp*

qed

lemma *perm-rstep-pos-conv* [simp]:

$\pi \cdot r \in \text{rstep-pos } R \ p \longleftrightarrow r \in \text{rstep-pos } R \ p$

by (*metis rstep-pos-eqv rstep-pos-permute trs-pt.mem-permute-iff*)

lemma *is-partition-map-vars-term-permute* [simp]:

$\text{is-partition } (\text{map } (\text{vars-term } \circ (\cdot) \ \pi) \ xs) \longleftrightarrow \text{is-partition } (\text{map vars-term } xs)$

by (*auto simp add: is-partition-def eqvt [symmetric]*)

lemma *linear-term-permute* [simp]:

$\text{linear-term } (\pi \cdot t) \longleftrightarrow \text{linear-term } t$

by (*induction t*) (*auto*)

lemma *ground-permute* [simp]:

$\text{ground } (\pi \cdot t) \longleftrightarrow \text{ground } t$

by (*induction t*) (*auto*)

lemma *funas-term-permute* [simp]:

$\text{funas-term } (\pi \cdot t) = \text{funas-term } t$

by (*induction t*) (*auto*)

lemma *in-NF-rstep-permute* [simp]:

$\pi \cdot t \in \text{NF } (\text{rstep } R) \longleftrightarrow t \in \text{NF } (\text{rstep } R)$

by (*metis NF-instance term-variants-iff*)

lemma *Id-trs-eqv* [simp]: $\pi \cdot (\text{Id} :: ('f, 'v :: \text{infinite}) \text{ trs}) = \text{Id}$

by (*auto simp: inv-rule-mem-trs-simps [where $p = \pi$, symmetric]*)

lemma *supteq-eqv* [simp]:

fixes $\pi :: ('v :: \text{infinite}) \text{ perm}$

shows $\pi \cdot \{\triangleright\} = \{\triangleright\}$

proof (*intro equalityI subrelI*)

fix $s \ t :: ('f, 'v) \text{ term}$ **assume** $(s, t) \in \pi \cdot \{\triangleright\}$

then have $-\pi \cdot s \triangleright -\pi \cdot t$ **by** *auto*

then have $(-\pi \cdot s) \cdot \text{sop } \pi \triangleright (-\pi \cdot t) \cdot \text{sop } \pi$ **by** *blast*

then show $s \triangleright t$ **by** *simp*

next

fix $s\ t :: ('f, 'v) \text{ term}$ **assume** $s \supseteq t$
from $\text{supteq-subst } [OF \text{ this, of sop } (-\pi)]$ **show** $(s, t) \in \pi \cdot \{\supseteq\}$ **by** simp
qed

lemma $\text{supt-eqvt } [\text{simp}]$:
 $\pi \cdot \{\supseteq\} = \{\supseteq\}$
by $(\text{simp add: supt-supteq-set-conv eqvt})$

lemma $\text{mgu-imp-mgu-var-disjoint}$:
fixes $v_x\ v_y :: 'v::\text{infinite} \Rightarrow 'v$ **and** $s\ t :: ('f, 'v) \text{ term}$
assumes $\mu: \text{mgu } s\ t = \text{Some } \mu$
and $\text{ren: inj } v_x\ \text{inj } v_y\ \text{range } v_x \cap \text{range } v_y = \{\}$
and $\text{disj: } V \cap W = \{\}$
and $\text{vars-term: vars-term } s \subseteq V\ \text{vars-term } t \subseteq W$
and $\text{finite: finite } V\ \text{finite } W$
shows $\exists \mu'$.
 $(\exists \pi.$
 $(\forall x \in V. \mu\ x = (\text{sop } \pi_1 \circ_s (\mu' \circ v_x) \circ_s \text{sop } (-\pi))\ x) \wedge$
 $(\forall x \in W. \mu\ x = (\text{sop } \pi_2 \circ_s (\mu' \circ v_y) \circ_s \text{sop } (-\pi))\ x)) \wedge$
 $\text{mgu } ((\pi_1 \cdot s) \cdot (\text{Var } \circ v_x))\ ((\pi_2 \cdot t) \cdot (\text{Var } \circ v_y)) = \text{Some } \mu' \text{ (is } \exists \mu'. - \wedge \text{mgu}$
 $\text{?sv ?tv} = -)$
proof –
define h **where** $h\ x = (\text{if } x \in V \text{ then } v_x\ (\text{Rep-perm } \pi_1\ x) \text{ else } v_y\ (\text{Rep-perm } \pi_2\ x))$ **for** x
have $\text{inj-on } h\ (V \cup W)$
using disj and ren
apply $(\text{unfold inj-on-def})$
by $(\text{auto simp: h-def dest!: injD } [OF \langle \text{inj } v_x \rangle]\ \text{injD } [OF \langle \text{inj } v_y \rangle]\ \text{split: if-splits})$
 $(\text{metis inv-Rep-perm-simp})+$
from $\text{inj-on-imp-bij-betw } [OF \text{ this}]$ **have** $\text{bij-betw } h\ (V \cup W)\ (h\ ' (V \cup W))$.
from $\text{bij-betw-extend } [OF \text{ this subset-UNIV subset-UNIV}]$
obtain f **where** $f \in \text{perms}$ **and** $f: \forall x \in V \cup W. f\ x = h\ x$
using $\text{finite by (auto simp: perms-def)}$
define π **and** μ' **where** $\pi = \text{Abs-perm } f$ **and** $\mu' = \text{sop } (-\pi) \circ_s \mu$
have $\text{finite (subst-domain } \mu)$
using $\text{mgu-subst-domain } [OF \mu]$ **and** $\text{finite and vars-term by (auto intro: finite-subset)}$
moreover **have** $\{x. \text{Rep-perm } (-\pi)\ x \neq x\} = \{x. \text{Rep-perm } \pi\ x \neq x\}$
using $\langle f \in \text{perms} \rangle$
by $(\text{auto simp: } \pi\text{-def perms-def})$
 $(\text{metis inv-Rep-perm-simp permute-atom-def})+$
moreover **then** **have** $\text{finite (subst-domain (Var } \circ \text{Rep-perm } (-\pi)))$
using $\langle f \in \text{perms} \rangle$ **by** $(\text{auto simp: subst-domain-def } \pi\text{-def Abs-perm-inverse perms-def})$
ultimately **have** $\text{finite: finite (subst-domain } \mu')$
by $(\text{auto simp: } \mu'\text{-def subst-domain-subst-compose})$

let $?s = \pi_1 \cdot s$ **and** $?t = \pi_2 \cdot t$

have $?sv = s \cdot \text{sop } \pi_1 \circ_s (\text{Var} \circ v_x)$ **by** *simp*
also have $\dots = s \cdot \text{sop } \pi$
using f **and** $\langle f \in \text{perms} \rangle$ **and** *vars-term* **and** *disj unfolding term-subst-eq-conv*
by (*auto simp: π -def h-def Abs-perm-inverse subst-compose*)
finally have $sv: ?sv = \pi \cdot s$ **by** *simp*

have $?tv = t \cdot \text{sop } \pi_2 \circ_s (\text{Var} \circ v_y)$ **by** *simp*
also have $\dots = t \cdot \text{sop } \pi$
using f **and** $\langle f \in \text{perms} \rangle$ **and** *vars-term* **and** *disj unfolding term-subst-eq-conv*
by (*auto simp: π -def h-def Abs-perm-inverse subst-compose split: if-splits*)
(metis Un-iff disjoint-iff-not-equal set-rev-mp)
finally have $tv: ?tv = \pi \cdot t$ **by** *simp*

have $eq: s \cdot \mu = t \cdot \mu$ **using** *mgu-sound* [*OF* $\langle \text{mgu } s \ t = \text{Some } \mu \rangle$] **by** (*auto simp: is-mgu-def*)
then have $eq': ?sv \cdot \mu' = ?tv \cdot \mu'$ **by** (*simp add: sv tv μ' -def*)

then obtain μ'' **where** $\mu'': \text{mgu } ?sv \ ?tv = \text{Some } \mu''$ **using** *mgu-complete* **by** (*auto simp: unifiers-def*)
from *mgu-sound* [*OF this*] **have** $eq'': ?sv \cdot \mu'' = ?tv \cdot \mu''$
and $\text{mgu}'': \text{is-mgu } \mu'' \{(?sv, ?tv)\}$ **by** (*auto simp: is-mgu-def is-mgu-def*)

{ fix $\tau :: (f, v)$ *subst* **assume** $*$: $?sv \cdot \tau = ?tv \cdot \tau$
then have $s \cdot \text{sop } \pi \circ_s \tau = t \cdot \text{sop } \pi \circ_s \tau$ **by** (*simp add: sv tv*)
with *mgu-sound* [*OF* μ , *THEN is-mgu-imp-is-mgu*] **obtain** δ **where** $\text{sop } \pi \circ_s \tau = \mu \circ_s \delta$
unfolding *is-mgu-def unifiers-def* **by** *force*
then have $\text{sop } (-\pi) \circ_s \text{sop } \pi \circ_s \tau = \text{sop } (-\pi) \circ_s \mu \circ_s \delta$ **by** (*auto simp: subst-compose-assoc*)
then have $\tau = \mu' \circ_s \delta$
by (*auto simp: μ' -def subst-compose-def*)
(metis atom-pt.permute-minus-cancel(1) permute-atom-def)
then have $\exists \gamma. \tau = \mu' \circ_s \gamma$ **by** *blast* **}**
then have $\text{is-mgu } \mu' \{(?sv, ?tv)\}$ **using** eq' **by** (*auto simp: is-mgu-def unifiers-def*)
with eq'' **obtain** γ **where** $\mu'' = \mu' \circ_s \gamma$ **by** (*auto simp: is-mgu-def unifiers-def*)
moreover obtain δ **where** $\mu' = \mu'' \circ_s \delta$ **using** eq' **and** mgu'' **by** (*auto simp: is-mgu-def unifiers-def*)
ultimately obtain π' **where** $*$: $\mu'' = \pi' \cdot \mu'$
using *subst-variants-imp-perm* [*OF* - - *finite mgu-finite-subst-domain* [*OF* μ''], *of γ δ*] **by** *metis*
have $\forall x \in V. \mu \ x = (\text{sop } \pi_1 \circ_s (\mu'' \circ v_x) \circ_s \text{sop } (-\pi')) \ x$
using f
by (*auto simp: $*$ μ' -def subst-compose-def permute-subst-def π -def h-def split: if-splits*)
(metis Abs-perm-inverse Rep-perm-uminus Un-iff $\langle f \in \text{perms} \rangle$ bij-Rep-perm bij-is-inj inv-f-eq)
moreover have $\forall x \in W. \mu \ x = (\text{sop } \pi_2 \circ_s (\mu'' \circ v_y) \circ_s \text{sop } (-\pi')) \ x$
using f **and** *disj*

```

    by (auto simp: *  $\mu'$ -def subst-compose-def permute-subst-def  $\pi$ -def h-def split:
if-splits)
    (metis Abs-perm-inverse Rep-perm-uminus Un-iff  $\langle f \in perms \rangle$  bij-Rep-perm
bij-is-inj disjoint-iff-not-equal inv-f-eq)
    ultimately show ?thesis using  $\mu''$  by blast
qed

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
  Renaming-Interpretations
begin

type-synonym ('f, 'v) rules = ('f, 'v) rule list

context fixes R :: ('f,'v)rules
begin

definition rrewrite :: ('f, 'v) term  $\Rightarrow$  ('f, 'v) term list
where
  rrewrite s = List.maps ( $\lambda (l, r) . \text{case match } s \text{ } l \text{ of}$ 
    None  $\Rightarrow$  []
    | Some  $\sigma \Rightarrow [r \cdot \sigma]$ ) R

lemma rrewrite-sound:  $t \in \text{set } (rrewrite \ s) \implies (s, t) \in rrstep \ (\text{set } R)$ 
  unfolding rrewrite-def List.maps-def using match-matches[of s]
  by force

lemma rrewrite-complete: assumes  $(s, t) \in rrstep \ (\text{set } R)$ 
  shows  $\exists u. u \in \text{set } (rrewrite \ s)$ 
proof -
  from assms obtain l r  $\sigma$  where lr:  $(l, r) \in \text{set } R$  and s:  $s = l \cdot \sigma$  and t:  $t = r$ 
  ·  $\sigma$ 
  by (rule rrstepE)
  from match-complete'[OF s[symmetric]] obtain  $\tau$  where match: match s l =
Some  $\tau$ 
  by auto
  with lr match have  $r \cdot \tau \in \text{set } (rrewrite \ s)$  unfolding rrewrite-def List.maps-def
  by force
  thus ?thesis ..
qed

```

```

lemma rewrite: assumes  $\bigwedge l\ r. (l,r) \in \text{set } R \implies \text{vars-term } l \supseteq \text{vars-term } r$ 
shows  $\text{set } (\text{rewrite } s) = \{t. (s,t) \in \text{rrstep } (\text{set } R)\}$ 
proof (standard; clarify)
  fix  $t$ 
  assume  $(s,t) \in \text{rrstep } (\text{set } R)$ 
  then obtain  $l\ r\ \sigma$  where  $lr: (l,r) \in \text{set } R$  and  $s: s = l \cdot \sigma$  and  $t: t = r \cdot \sigma$ 
  by (rule rrstepE)
  from match-complete'[OF s[symmetric]] obtain  $\tau$  where match: match s l =
Some  $\tau$ 
  and  $\text{vars}: \bigwedge x. x \in \text{vars-term } l \implies \sigma\ x = \tau\ x$  by auto
  have  $\text{vars}': \bigwedge x. x \in \text{vars-term } r \implies \sigma\ x = \tau\ x$  using assms[OF lr]  $\text{vars}$  by
auto
  have  $t: t = r \cdot \tau$  unfolding  $t$  using  $\text{vars}'$  by (intro term-subst-eq, auto)
  with  $lr$  match show  $t \in \text{set } (\text{rewrite } s)$  unfolding rewrite-def List.maps-def
by force
qed (rule rewrite-sound)

fun rewrite :: (f, v) term  $\Rightarrow$  (f, v) term list where
  rewrite s = (rrrewrite s @ (case s of Var -  $\Rightarrow$  [] | Fun f ss  $\Rightarrow$ 
    concat (map ( $\lambda (i, si). \text{map } (\lambda ti. \text{Fun } f\ (ss[i := ti]))\ (\text{rewrite } si)$ )
      (zip [ $0..<\text{length } ss$ ] ss))))))

declare rewrite.simps[simp del]

lemma rewrite-sound:  $t \in \text{set } (\text{rewrite } s) \implies (s,t) \in \text{rstep } (\text{set } R)$ 
proof (induct s arbitrary: t rule: rewrite.induct)
  case (1 s t)
  note [simp] = rewrite.simps[of s]
  from 1(2) consider (root)  $t \in \text{set } (\text{rrrewrite } s)$  |
    (arg)  $f\ ss\ ti\ i$  where  $s = \text{Fun } f\ ss\ i < \text{length } ss\ ti \in \text{set } (\text{rewrite } (ss ! i))\ t =$ 
Fun f (ss[i := ti])
  by (auto simp: set-zip)
  thus ?case
proof cases
  case root
  with rewrite-sound[of t s] have  $(s,t) \in \text{rrstep } (\text{set } R)$  by auto
  thus ?thesis by (rule rrstep-imp-rstep)
next
  case (arg f ss ti i)
  from arg(2) have  $\text{mem}: (i, ss ! i) \in \text{set } (\text{zip } [0..<\text{length } ss]\ ss)$  by (force simp:
set-zip)
  from 1(1)[OF arg(1) mem refl arg(3)]
  have IH:  $(ss ! i, ti) \in \text{rstep } (\text{set } R)$  .
  with arg have  $(s,t) \in \text{rrstep } (\text{set } R)$ 
  unfolding rrstep-iff-arg-rstep by blast
  thus ?thesis by (rule nrrstep-imp-rstep)
qed
qed

```

```

lemma rewrite: assumes  $\bigwedge l\ r. (l,r) \in \text{set } R \implies \text{vars-term } l \supseteq \text{vars-term } r$ 
  shows  $\text{set } (\text{rewrite } s) = \{t. (s,t) \in \text{rstep } (\text{set } R)\}$ 
proof (standard; clarify)
  fix  $t$ 
  assume  $(s,t) \in \text{rstep } (\text{set } R)$ 
  then obtain  $C\ u\ v$  where  $s: s = C\langle u \rangle$  and  $t: t = C\langle v \rangle$  and  $\text{step}: (u,v) \in \text{rrstep}$ 
  ( $\text{set } R$ )
    by blast
  from  $\text{rrewrite}[OF\ \text{assms},\ \text{of } u]\ \text{step}$  have  $\text{step}: v \in \text{set } (\text{rrewrite } u)$  by auto
  show  $t \in \text{set } (\text{rewrite } s)$  unfolding  $s\ t$ 
  proof (induct C)
    case Hole
    then show  $?case$  using  $\text{step}$  by (auto simp: rewrite.simps[of u])
  next
    case ( $\text{More } f\ \text{bef } C\ \text{aft}$ )
    show  $?case$ 
      apply (simp add: rewrite.simps[of Fun f -] set-zip)
      apply (intro disjI2)
      apply (intro exI[of - C⟨u⟩] exI)
      apply (intro conjI exI[of - length bef])
      using More by (auto simp: nth-append)
    qed
  qed (rule rewrite-sound)

lemma rewrite-complete: assumes  $(s,t) \in \text{rstep } (\text{set } R)$ 
  shows  $\exists w. w \in \text{set } (\text{rewrite } s)$ 
proof –
  from assms obtain  $C\ u\ v$  where  $s: s = C\langle u \rangle$  and  $t: t = C\langle v \rangle$  and  $\text{step}: (u,v)$ 
   $\in \text{rrstep } (\text{set } R)$ 
    by blast
  from  $\text{rrewrite-complete}[OF\ \text{step}]$  obtain  $v$  where  $\text{step}: v \in \text{set } (\text{rrewrite } u)$  by
  auto
  have  $C\langle v \rangle \in \text{set } (\text{rewrite } s)$  unfolding  $s$ 
  proof (induct C)
    case Hole
    then show  $?case$  using  $\text{step}$  by (auto simp: rewrite.simps[of u])
  next
    case ( $\text{More } f\ \text{bef } C\ \text{aft}$ )
    show  $?case$ 
      apply (simp add: rewrite.simps[of Fun f -] set-zip)
      apply (intro disjI2)
      apply (intro exI[of - C⟨u⟩] exI)
      apply (intro conjI exI[of - length bef])
      using More by (auto simp: nth-append)
    qed
  qed
  thus  $?thesis$  by blast
qed
end

```

lemma *rewrite-mono*: $set\ R \subseteq set\ S \implies set\ (rewrite\ R\ s) \subseteq set\ (rewrite\ S\ s)$
unfolding *rewrite-def List.maps-def* **by** *auto*

lemma *Union-image-mono*: $(\bigwedge x. x \in A \implies f\ x \subseteq g\ x) \implies \bigcup (f\ ` A) \subseteq \bigcup (g\ ` A)$
by *blast*

lemma *rewrite-mono*: **assumes** $set\ R \subseteq set\ S$
shows $set\ (rewrite\ R\ s) \subseteq set\ (rewrite\ S\ s)$
proof –
note $rewrite = rewrite-mono[OF\ assms]$
show *?thesis*
proof (*induct s*)
case (*Var x*)
thus *?case* **using** *rewrite* **unfolding** *rewrite.simps[of - Var x]* **by** *auto*
next
case (*Fun f ss*)
show *?case* **unfolding** *rewrite.simps[of - Fun f ss]*
 $set-append\ term.simps\ set-concat\ set-map\ image-comp\ set-zip\ o-def$
apply (*intro Un-mono, rule rewrite*)
by (*intro Union-image-mono, insert Fun, force simp: set-conv-nth[of ss]*)
qed
qed

definition *first-rewrite* :: $(f, v)rules \Rightarrow (f, v)term \Rightarrow (f, v)term\ option$
where $first-rewrite\ R\ s \equiv case\ rewrite\ R\ s\ of\ Nil \Rightarrow None \mid Cons\ t\ - \Rightarrow Some\ t$

definition *is-root-step* :: $(f, v)trs \Rightarrow (f, v)\ term \Rightarrow (f, v)\ term \Rightarrow bool$
where
 $is-root-step\ R\ s\ t = (\exists\ (l, r) \in R. case\ match-list\ Var\ [(l, s), (r, t)]\ of$
 $None \Rightarrow False$
 $\mid Some\ - \Rightarrow True)$

lemma *rrstep-code[code-unfold]*: $(s, t) \in rrstep\ R \longleftrightarrow is-root-step\ R\ s\ t$
proof
show $is-root-step\ R\ s\ t \implies (s, t) \in rrstep\ R$
unfolding *is-root-step-def rrstep-def rstep-r-p-s-def'*
by (*auto split: option.splits*) (*force dest: match-list-matches*)
assume $(s, t) \in rrstep\ R$
then obtain $\sigma\ l\ r$ **where** $lr: (l, r) \in R$ **and** $id: s = l \cdot \sigma\ t = r \cdot \sigma$
by (*metis rrstepE*)
show $is-root-step\ R\ s\ t$ **unfolding** *id*
unfolding *is-root-step-def*
by (*cases match-list Var [(l, l · σ), (r, r · σ)],*
 $auto\ intro!:\ bexI[OF\ -\ lr]\ dest!:\ match-list-complete$)

qed

lemma *is-root-step*: *is-root-step* R s $t \implies (s, t) \in rstep\ R$
unfolding *rrstep-code* .

fun *is-rstep* :: $(f, 'v)trs \Rightarrow (f, 'v)term \Rightarrow (f, 'v)term \Rightarrow bool$ **where**
is-rstep R $(Fun\ f\ ts)$ $(Fun\ g\ ss) =$ (
 $f = g \wedge length\ ts = length\ ss \wedge (\exists\ i \in set\ [0..<length\ ss].$
 $ss = ts[i := ss\ !\ i] \wedge is-rstep\ R\ (ts\ !\ i)\ (ss\ !\ i))$
 $\vee (Fun\ f\ ts, Fun\ g\ ss) \in rstep\ R)$
| *is-rstep* $R\ s\ t = ((s, t) \in rstep\ R)$

lemma *is-rstep-sound*: *is-rstep* R $s\ t \implies (s, t) \in rstep\ R$

proof (*induct* $R\ s\ t$ *rule*: *is-rstep.induct*)
case $(1\ R\ f\ ts\ g\ ss)$
show ?*case*
proof (*cases* $(Fun\ f\ ts, Fun\ g\ ss) \in rstep\ R$)
case *True*
thus ?*thesis* **using** *rrstep-imp-rstep* **by** *auto*
next
case *False*
with $1(2)$ **obtain** i **where**
 $i: i < length\ ss$ **and**
 $gf: g = f$ **and** $len: length\ ts = length\ ss$ **and** $id: ss = ts[i := ss\ !\ i]$
and $rec: is-rstep\ R\ (ts\ !\ i)\ (ss\ !\ i)$
by *auto*
from $1(1)[OF - rec]\ i$ **have** $(ts\ !\ i, ss\ !\ i) \in rstep\ R$ **by** *auto*
thus ?*thesis* **unfolding** gf **using** $id\ i\ len$
by (*metis* *nrrstep-iff-arg-rstep* *nrrstep-imp-rstep*)
qed
qed (*insert* *rrstep-imp-rstep*, *auto*)

lemma *is-rstep-complete*: **assumes** $(s, t) \in rstep\ R$

shows *is-rstep* $R\ s\ t$

proof –

from *rstepE*[*OF* *assms*] **obtain** $C\ s'\ t'$ **where**
 $id: s = C\ \langle s' \rangle\ t = C\ \langle t' \rangle$ **and** $step: (s', t') \in rstep\ R$
using *rrstepI* **by** *metis*
show ?*thesis* **unfolding** id
proof (*induct* C)
case *Hole*
then show ?*case* **using** *step* **by** (*cases* s' ; *cases* t' , *auto*)
next
case $(More\ f\ bef\ C\ aft)$
show ?*case* **unfolding** *ctxt-apply-term.simps* *is-rstep.simps*
by (*intro* *disjI1* *conjI* *bexI*[*of* - *length* *bef*], *insert* *More*, *auto*)
qed
qed

lemma *is-rstep[simp]*: *is-rstep* *R s t* \longleftrightarrow (*s,t*) \in *rstep R*
using *is-rstep-sound is-rstep-complete* **by** *auto*

lemma *in-rstep-code[code-unfold]*:
 $st \in rstep\ R \longleftrightarrow (case\ st\ of\ (s,t) \Rightarrow is-rstep\ R\ s\ t)$
by (*cases st, auto*)

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)*)^{*} **using** *res[unfolded compute-rstep-NF-def]*
proof (*rule compute-NF-sound*)
fix *s t*
assume *first-rewrite R s* = *Some t*
from *this[unfolded first-rewrite-def]* **obtain** *ts* **where** *rewrite R s* = *t # ts*
by (*cases rewrite R s, auto*)
then have *t*: *t* \in *set (rewrite R s)* **by** *simp*
from *rewrite-sound[OF this]* **show** (*s,t*) \in *rstep (set R)* .
qed

lemma *compute-rstep-NF-complete*: **assumes** *res*: *compute-rstep-NF R s* = *Some t*
shows *t* \in *NF (rstep (set R))* **using** *res[unfolded compute-rstep-NF-def]*
proof (*rule compute-NF-complete*)
fix *s*
assume *first-rewrite R s* = *None*
from *this[unfolded first-rewrite-def]* **have** *empty*: *rewrite R s* = []
by (*cases rewrite R s, auto*)
have *False* **if** (*s,t*) \in *rstep (set R)* **for** *t*
using *rewrite-complete[OF that] arg-cong[OF empty, of set]* **by** *auto*
thus *s* \in *NF (rstep (set R))* **by** *blast*
qed

lemma *compute-rstep-NF-SN*: **assumes** *SN*: *SN (rstep (set R))*
shows $\exists\ t.$ *compute-rstep-NF R s* = *Some t*
proof –
have $\exists\ t.$ *compute-NF (first-rewrite R) s* = *Some t*
proof (*rule compute-NF-SN[OF SN]*)
fix *s t*
assume *first-rewrite R s* = *Some t*
from *this[unfolded first-rewrite-def]* **have**
 $rewrite: t \in set\ (rewrite\ R\ s)$ **by** (*auto split: list.splits*)
from *rewrite-sound[OF this]*
show (*s,t*) \in *rstep (set R)* .
qed
then show *?thesis* **unfolding** *compute-rstep-NF-def* .
qed

definition

```

check-join-NF ::
  ('f :: showl, 'v :: showl) rules =>
  ('f, 'v) term => ('f, 'v) term => showsl check
where
  check-join-NF R s t ≡ case (compute-rstep-NF R s, compute-rstep-NF R t) of
    (Some s', Some t') =>
      check (s' = t') (
        showsl (STR "the normal form '" ∘ showsl s' ∘ showsl (STR " of '" ∘ showsl
s
          ∘ showsl (STR " differs from  $\Leftarrow$  the normal form '" ∘ showsl t' ∘ showsl (STR
" of '" ∘ showsl t)
        | - => 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) ∈ join (rstep (set R))
proof -
  note ok = ok[unfolded check-join-NF-def]
  from ok obtain s' where s: compute-rstep-NF R s = Some s' by force
  note ok = ok[unfolded s]
  from ok obtain t' where t: compute-rstep-NF R t = Some t' by force
  from ok[unfolded t] have id: s' = t' by simp
  note seq = compute-rstep-NF-sound
  from seq[OF s] seq[OF t]
  show ?thesis unfolding id by auto
qed

```

fun *reachable-terms* ::

```

('f, 'v) rules => ('f, 'v) term => nat => ('f, 'v) term list
where
  reachable-terms R s 0 = [s]
| reachable-terms R s (Suc n) = (
  let ts = (reachable-terms R s n) in
  remdups (ts@(concat (map (λ t. rewrite R t) ts)))
)

```

lemma *reachable-terms-nat*:

```

assumes t ∈ set (reachable-terms R s i)
shows ∃ j. j ≤ i ∧ (s, t) ∈ (rstep (set R))~j
using assms
proof (induct i arbitrary: t)
  case 0
  then show ?case by auto
next
  case (Suc i)
  let ?R = λ j. (rstep (set R))~j
  from Suc(2)

```

```

have  $t \in \text{set } (\text{reachable-terms } R \ s \ i)$ 
   $\vee (\exists \ u \in \text{set } (\text{reachable-terms } R \ s \ i). \ t \in \text{set } (\text{rewrite } R \ u))$  by (simp add:
Let-def)
then show ?case
proof
  assume  $t \in \text{set } (\text{reachable-terms } R \ s \ i)$ 
  from  $\text{Suc}(1)[\text{OF } \text{this}]$  obtain  $j$  where  $j \leq i$  and  $(s,t) \in ?R \ j$  by auto
  then show ?thesis by (intro exI[of - j], auto)
next
  assume  $\exists \ u \in \text{set } (\text{reachable-terms } R \ s \ i). \ t \in \text{set } (\text{rewrite } R \ u)$ 
  then obtain  $u$  where  $u: u \in \text{set } (\text{reachable-terms } R \ s \ i)$ 
    and  $1: t \in \text{set } (\text{rewrite } R \ u)$  by auto
  from rewrite-sound[OF 1] have  $ut: (u,t) \in \text{rstep } (\text{set } R)$  .
  from  $\text{Suc}(1)[\text{OF } u]$  obtain  $j$  where  $j: j \leq i$  and  $su: (s,u) \in ?R \ j$  by auto
  from  $su \ ut$  have  $(s,t) \in ?R \ (\text{Suc } j)$  by auto
  with  $j$  show ?thesis by (intro exI[of - Suc j], auto)
qed
qed

```

```

lemma reachable-terms:
assumes  $t \in \text{set } (\text{reachable-terms } R \ s \ i)$ 
shows  $(s,t) \in (\text{rstep } (\text{set } R))^*$ 
  using reachable-terms-nat[OF assms] by (metis relpow-imp-rtrancl)

```

```

lemma reachable-terms-one:
assumes  $t \in \text{set } (\text{reachable-terms } R \ s \ (\text{Suc } 0))$ 
shows  $(s,t) \in (\text{rstep } (\text{set } R))^1$ 
proof -
  from reachable-terms-nat[OF assms] obtain  $j$  where  $j \leq 1$ 
  and  $(s,t) \in (\text{rstep } (\text{set } R))^j$  by auto
  then show ?thesis by (cases j, auto)
qed

```

```

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
  iterative-join-search-main  $R \ s \ t \ i \ n = (\text{if } i \leq n \text{ then}$ 
     $((\text{list-inter } (\text{reachable-terms } R \ s \ i) \ (\text{reachable-terms } R \ t \ i)) \neq []) \vee (\text{iterative-join-search-main}$ 
 $R \ s \ t \ (\text{Suc } i) \ n)) \text{ else False})$ 
by pat-completeness auto

```

```

termination by (relation measure  $(\lambda (R,s,t,i,n). \text{Suc } n - i)$ ) auto

```

```

lemma iterative-join-search-main:
  iterative-join-search-main  $R \ s \ t \ i \ n \Longrightarrow (s,t) \in \text{join } (\text{rstep } (\text{set } R))$ 
proof (induction rule: iterative-join-search-main.induct)
  case  $(1 \ R \ s \ t \ i \ n)$ 
  from  $1(2)$  have  $i-n: i \leq n$  by (simp split: if-splits)

```

```

note  $IH = 1(1)[OF\ i-n]$ 
let  $?I = list-inter\ (reachable-terms\ R\ s\ i)\ (reachable-terms\ R\ t\ i)$ 
from  $1(2)\ i-n$  have  $?I \neq [] \vee iterative-join-search-main\ R\ s\ t\ (Suc\ i)\ n$  by auto
then show  $?case$ 
proof
  assume  $a: ?I \neq []$ 
  then obtain  $u\ us$  where  $u: ?I = u \# us$  by  $(cases\ ?I,\ auto)$ 
  then have  $d: u \in set\ ?I$  by auto
  from  $this[simplified]\ reachable-terms[of\ u - -\ i]$  have  $c: (s, u) \in (rstep\ (set\ R))^*$ 
by auto
  from  $d[simplified]\ reachable-terms[of\ u - -\ i]$  have  $e: (t, u) \in (rstep\ (set\ R))^{\wedge*}$ 
by auto
  from  $c\ e$  show  $?thesis$  by auto
next
  assume  $b: iterative-join-search-main\ R\ s\ t\ (Suc\ i)\ n$ 
  from  $IH[OF\ this]$  show  $?thesis$  .
qed
qed

```

definition *iterative-join-search* **where**

$iterative-join-search\ R\ s\ t\ n \equiv iterative-join-search-main\ R\ s\ t\ 0\ n$

lemma *iterative-join-search*: $iterative-join-search\ R\ s\ t\ n \implies (s, t) \in join\ (rstep\ (set\ R))$

by $(rule\ iterative-join-search-main,\ unfold\ iterative-join-search-def)$

definition

$check-join-BFS-limit ::$
 $nat \Rightarrow ('f :: showsl,\ 'v :: showsl)\ rules \Rightarrow$
 $('f,\ 'v)\ term \Rightarrow ('f,\ 'v)\ term \Rightarrow showsl\ check$

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 $(s, t) \in join\ (rstep\ (set\ R))$

by $(rule\ iterative-join-search,\ insert\ ok[unfolded\ check-join-BFS-limit-def],\ simp)$

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* $[simp]$:

$set\ (map-funs-rules\ fg\ R) = map-funs-trs\ fg\ (set\ R)$

unfolding *map-funs-rules-def* *map-funs-trs.simps* **by** *simp*

16.1 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* " \rightarrow ")

definition *showsl-weak-rule* \equiv *showsl-rule'* *showsl* *showsl* (*STR* " \rightarrow = ")

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* " \leftrightarrow ")
(*STR* ""') trs \circ *showsl-nl*

definition *showsl-rules* \equiv *showsl-rules'* *showsl* *showsl* (*STR* " \rightarrow ")

definition *showsl-weak-rules* \equiv *showsl-rules'* *showsl* *showsl* (*STR* " \rightarrow = ")

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* " \leftrightarrow " \leftrightarrow ") \circ
showsl-rules' fun var arr R

definition *showsl-trs* \equiv *showsl-trs'* *showsl* *showsl* (*STR* "rewrite system:") (*STR* " \rightarrow ")

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

where

add-vars-rule r xs = *add-vars-term* (fst r) (*add-vars-term* (snd r) xs)

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

where

add-funs-rule r fs = *add-funs-term* (fst r) (*add-funs-term* (snd r) fs)

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

where

add-funas-rule r fs = *add-funas-term* (fst r) (*add-funas-term* (snd r) fs)

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

where

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

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

where

$add_funas_args_rule\ r\ fs = add_funas_args_term\ (fst\ r)\ (add_funas_args_term\ (snd\ r)\ fs)$

lemma *add-vars-rule-vars-rule-list-conv* [simp]:
 $add_vars_rule\ r\ xs = vars_rule_list\ r\ @\ xs$
by (simp add: add-vars-rule-def vars-rule-list-def)

lemma *add-funs-rule-funs-rule-list-conv* [simp]:
 $add_funs_rule\ r\ fs = funs_rule_list\ r\ @\ fs$
by (simp add: add-funs-rule-def funs-rule-list-def)

lemma *add-funas-rule-funas-rule-list-conv* [simp]:
 $add_funas_rule\ r\ fs = funas_rule_list\ r\ @\ fs$
by (simp add: add-funas-rule-def funas-rule-list-def)

lemma *add-roots-rule-roots-rule-list-conv* [simp]:
 $add_roots_rule\ r\ fs = roots_rule_list\ r\ @\ fs$
by (simp add: add-roots-rule-def roots-rule-list-def)

lemma *add-funas-args-rule-funas-args-rule-list-conv* [simp]:
 $add_funas_args_rule\ r\ fs = funas_args_rule_list\ r\ @\ fs$
by (simp add: add-funas-args-rule-def funas-args-rule-list-def)

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

$insert_vars_rule\ r\ xs = insert_vars_term\ (fst\ r)\ (insert_vars_term\ (snd\ r)\ xs)$

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

$insert_funs_rule\ r\ fs = insert_funs_term\ (fst\ r)\ (insert_funs_term\ (snd\ r)\ fs)$

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

$insert_funas_rule\ r\ fs = insert_funas_term\ (fst\ r)\ (insert_funas_term\ (snd\ r)\ fs)$

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

$insert_roots_rule\ r\ fs =$
 $foldr\ List.insert\ (option-to-list\ (root\ (fst\ r))\ @\ option-to-list\ (root\ (snd\ r)))\ fs$

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

$insert_funas_args_rule\ r\ fs = insert_funas_args_term\ (fst\ r)\ (insert_funas_args_term\ (snd\ r)\ fs)$

lemma *set-insert-vars-rule* [simp]:
 $set\ (insert_vars_rule\ r\ xs) = vars_term\ (fst\ r) \cup vars_term\ (snd\ r) \cup set\ xs$
by (simp add: insert-vars-rule-def ac-simps)

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

set (insert-funs-rule r xs) = funs-term (fst r) \cup funs-term (snd r) \cup set xs

by (simp add: insert-funs-rule-def ac-simps)

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

set (insert-funas-rule r xs) = funas-term (fst r) \cup funas-term (snd r) \cup set xs

by (simp add: insert-funas-rule-def ac-simps)

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

set (insert-roots-rule r xs) = root-set (fst r) \cup root-set (snd r) \cup set xs

by (cases fst r snd r rule: term.exhaust [case-product term.exhaust])

(auto simp: insert-roots-rule-def ac-simps)

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

set (insert-funas-args-rule r xs) = funas-args-term (fst r) \cup funas-args-term (snd r) \cup set xs

by (simp add: insert-funas-args-rule-def ac-simps funas-args-term-def)

abbreviation *vars-rule-impl* $r \equiv$ insert-vars-rule r []

abbreviation *funs-rule-impl* $r \equiv$ insert-funs-rule r []

abbreviation *funas-rule-impl* $r \equiv$ insert-funas-rule r []

abbreviation *roots-rule-impl* $r \equiv$ insert-roots-rule r []

abbreviation *funas-args-rule-impl* $r \equiv$ insert-funas-args-rule r []

lemma *set-vars-rule-impl*:

set (vars-rule-impl r) = vars-rule r

by (simp add: vars-rule-def)

lemma [code]:

vars-rule-list r = add-vars-rule r []

funs-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 []

by (simp-all add: vars-rule-list-def funs-rule-list-def funas-rule-list-def
roots-rule-list-def funas-args-rule-list-def)

lemma [code]:

vars-trs-list trs = foldr add-vars-rule trs []

funs-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 []

by (induct trs)

(simp-all add: vars-trs-list-def funs-trs-list-def funas-trs-list-def
roots-trs-list-def funas-args-trs-list-def)

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 $\text{insert-funs-trs} :: ('f, 'v) \text{ rule list} \Rightarrow 'f \text{ list} \Rightarrow 'f \text{ list}$

where

$\text{insert-funs-trs trs} = \text{foldr insert-funs-rule trs}$

definition $\text{insert-funas-trs} :: ('f, 'v) \text{ rule list} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{insert-funas-trs trs} = \text{foldr insert-funas-rule trs}$

definition $\text{insert-roots-trs} :: ('f, 'v) \text{ rule list} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{insert-roots-trs trs} = \text{foldr insert-roots-rule trs}$

definition $\text{insert-funas-args-trs} :: ('f, 'v) \text{ rule list} \Rightarrow ('f \times \text{nat}) \text{ list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{insert-funas-args-trs trs} = \text{foldr insert-funas-args-rule trs}$

lemma $\text{set-insert-vars-trs [simp]}:$

$\text{set} (\text{insert-vars-trs trs xs}) = (\bigcup r \in \text{set trs. vars-rule } r) \cup \text{set xs}$

by $(\text{induct trs arbitrary: xs}) (\text{simp-all add: insert-vars-trs-def ac-simps vars-rule-def})$

lemma $\text{set-insert-funs-trs [simp]}:$

$\text{set} (\text{insert-funs-trs trs fs}) = (\bigcup r \in \text{set trs. funs-rule } r) \cup \text{set fs}$

by $(\text{induct trs arbitrary: fs}) (\text{simp-all add: insert-funs-trs-def ac-simps funs-rule-def})$

lemma $\text{set-insert-funas-trs [simp]}:$

$\text{set} (\text{insert-funas-trs trs fs}) = (\bigcup r \in \text{set trs. funas-rule } r) \cup \text{set fs}$

by $(\text{induct trs arbitrary: fs}) (\text{simp-all add: insert-funas-trs-def ac-simps funas-rule-def})$

lemma $\text{set-insert-roots-trs [simp]}:$

$\text{set} (\text{insert-roots-trs trs fs}) = (\bigcup r \in \text{set trs. roots-rule } r) \cup \text{set fs}$

by $(\text{induct trs arbitrary: fs}) (\text{simp-all add: insert-roots-trs-def ac-simps roots-rule-def})$

lemma $\text{set-insert-funas-args-trs [simp]}:$

$\text{set} (\text{insert-funas-args-trs trs fs}) = (\bigcup r \in \text{set trs. funas-args-rule } r) \cup \text{set fs}$

by $(\text{induct trs arbitrary: fs})$

$(\text{simp-all add: insert-funas-args-trs-def ac-simps funas-args-rule-def})$

abbreviation $\text{vars-trs-impl trs} \equiv \text{insert-vars-trs trs } []$

abbreviation $\text{funs-trs-impl trs} \equiv \text{insert-funs-trs trs } []$

abbreviation $\text{funas-trs-impl trs} \equiv \text{insert-funas-trs trs } []$

abbreviation $\text{roots-trs-impl trs} \equiv \text{insert-roots-trs trs } []$

abbreviation $\text{funas-args-trs-impl trs} \equiv \text{insert-funas-args-trs trs } []$

definition $\text{defined-list} :: ('f, 'v) \text{ rule list} \Rightarrow ('f \times \text{nat}) \text{ list}$

where

$\text{defined-list } R = [\text{the (root } l). (l, r) \leftarrow R, \text{ is-Fun } l]$

lemma *set-defined-list* [simp]:
 $set\ (defined-list\ R) = \{fn.\ defined\ (set\ R)\ fn\}$
by (force simp: defined-list-def defined-def elim!: root-Some)

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]:
 $isOK\ (check-left-linear-trs\ R) = left-linear-trs\ (set\ R)$
unfolding *check-left-linear-trs-def* *left-linear-trs-def*
by auto

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]:
 $isOK\ (check-varcond-subset\ R) = (\forall\ (l, r) \in set\ R.\ vars-term\ r \subseteq vars-term\ l)$
unfolding *check-varcond-subset-def* **by** force+

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]:
 $isOK\ (check-varcond-no-Var-lhs\ R) \longleftrightarrow (\forall\ (l, r) \in set\ R.\ is-Fun\ l)$
by (auto simp: check-varcond-no-Var-lhs-def)

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]:
 $isOK\ (check-wf-trs\ R) = wf-trs\ (set\ R)$
by (force simp: check-wf-trs-def wf-trs-def)

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 –

from *assms* **have** $\neg wf-trs\ (set\ R)$ **unfolding** *check-not-wf-trs-def* **by** *auto*

with *SN-rstep-imp-wf-trs* **show** *?thesis* **by** *auto*

qed

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

assume *?l*

then obtain σ **where** $fst\ lr = fst\ st \cdot \sigma$

and $snd\ lr = snd\ st \cdot \sigma$ **by** (*auto simp: instance-rule-def*)

then have $\bigwedge l\ t. (l,\ t) \in set\ ?list \implies l \cdot \sigma = t$ **by** (*auto*)

then have $matchers\ (set\ ?list) \neq \{\}$ **by** (*auto simp: matchers-def*)

with *match-list-complete*

show $match-list\ ?d\ ?list \neq None$ **by** *blast*

next

assume $match-list\ ?d\ ?list \neq None$

then obtain τ **where** $match-list\ ?d\ ?list = Some\ \tau$ **by** *auto*

from *match-list-sound* [*OF this*]

have $fst\ lr = fst\ st \cdot \tau$ **and** $snd\ lr = snd\ st \cdot \tau$ **by** *auto*

then show *?l* **by** (*auto simp: instance-rule-def*)

qed

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

assume *?l*

show *?r*

proof

fix $x\ y$

assume $(x,y) \in subst.closure\ (set\ R)$

then show $(x,y) \in subst.closure\ (set\ S)$

proof (*induct*)

case ($subst\ s\ t\ \sigma$)

```

with ⟨?l⟩[unfolded check-CS-subseteq-def]
obtain  $l\ r\ \delta$  where  $lr: (l,r) \in \text{set } S$  and  $s: s = l \cdot \delta$  and  $t: t = r \cdot \delta$ 
  by (auto simp add: instance-rule-def)
show ?case unfolding  $s\ t$ 
  using subst.closure.intros[OF  $lr$ , of  $\delta \circ_s \sigma$ ]
  by auto
qed
qed
next
  assume ?r
  {
    fix  $lr$ 
    assume mem:  $lr \in \text{set } R$ 
    obtain  $l\ r$  where  $lr: lr = (l,r)$  by (cases  $lr$ , auto)
    with mem have  $(l,r) \in \text{subst.closure } (\text{set } R)$  using subst.subset-closure by
  auto
    with ⟨?r⟩ have  $(l,r) \in \text{subst.closure } (\text{set } S)$  by auto
    then have Bex (set  $S$ ) (instance-rule  $lr$ ) unfolding  $lr$ 
    proof (induct)
      case (subst  $s\ t\ \sigma$ )
      then show ?case unfolding instance-rule-def by force
    qed
  }
  thus ?l unfolding check-CS-subseteq-def by auto
qed

```

type-synonym $(f, 'v)\text{rule-map} = ((f \times \text{nat}) \Rightarrow (f, 'v)\text{rules})\text{option}$

```

fun computeRuleMapH ::  $(f, 'v)\text{rules} \Rightarrow ((f \times \text{nat}) \times (f, 'v)\text{rules})\text{list option}$ 
where computeRuleMapH [] = Some []
  | computeRuleMapH ((Fun  $f\ ts, r$ ) # rules) = (let  $n = \text{length } ts$  in case com-
puteRuleMapH rules of None  $\Rightarrow$  None | Some  $rm \Rightarrow$ 
    (case List.extract  $(\lambda (fa, rls). fa = (f, n))\ rm$  of
      None  $\Rightarrow$  Some  $((f, n), [(Fun\ f\ ts, r)])$  #  $rm$ 
    | Some  $(bef, (fa, rls), aft) \Rightarrow$  Some  $((fa, (Fun\ f\ ts, r)) \# rls) \# bef @$ 
aft)))
  | computeRuleMapH ((Var -, -) # rules) = None

```

definition computeRuleMap :: $(f, 'v)\text{rules} \Rightarrow (f, 'v)\text{rule-map}$ **where**
 computeRuleMap $rls \equiv$
 (case computeRuleMapH rls of
 None \Rightarrow None
 | Some $rm \Rightarrow$ Some $(\lambda f.$
 (case map-of $rm\ f$ of
 None \Rightarrow []
 | Some $rls \Rightarrow rls)))$

```

lemma computeRuleMapHSound2: (computeRuleMapH R = None) = ( $\exists$  (l, r)  $\in$ 
set R. root l = None)
proof (induct R)
  case (Cons rule rules)
  obtain l r where rule: rule = (l,r) by force
  show ?case
  proof (cases l)
    case (Fun f ts)
    show ?thesis
    using rule Cons
  proof (cases computeRuleMapH rules)
    case (Some rm) note oSome = this
    let ?e = List.extract ( $\lambda$  (fa,rls). fa = (f,length ts)) rm
    from Some Fun rule Cons show ?thesis
    proof (cases ?e)
      case (Some res)
      then obtain bef aft g rls where ?e = Some (bef, (g,rls), aft) by (cases res,
force)
      with extract-SomeE[OF this] have rm: rm = bef @ ((f, length ts),rls) #
aft and e: ?e = Some (bef, ((f,length ts),rls), aft)
      by auto
      show ?thesis using Cons
      by (simp add: rule Fun Let-def oSome e)
    qed auto
    qed (insert, auto simp: rule Fun)
    qed (auto simp: rule)
  qed (auto simp: rule)

```

```

lemma computeRuleMapSound2: (computeRuleMap R = None) = ( $\exists$  (l, r)  $\in$  set
R. root l = None)
unfolding computeRuleMap-def
by (simp only: computeRuleMapHSound2[symmetric], cases computeRuleMapH R,
auto)

```

```

lemma computeRuleMapHSound: assumes computeRuleMapH R = Some rm
  shows ( $\lambda$  (f,rls). (f,set rls)) ‘ set rm = {((f,n),{(l,r) | l r. (l,r)  $\in$  set R  $\wedge$  root l
= Some (f, n)}) | f n. {(l,r) | 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
using assms
proof (induct R arbitrary: rm)
  case (Cons rule rules)
  let ?setl =  $\lambda$  rm. ( $\lambda$  (f,rls). (f,set rls)) ‘ set rm
  let ?setr =  $\lambda$  R. {(f,n),{(l,r) | l r. (l,r)  $\in$  set R  $\wedge$  root l = Some (f, n)}} | f n.
{(l,r) | l r. (l,r)  $\in$  set R  $\wedge$  root l = Some (f, n)}  $\neq$  {}}
  obtain l r where Pair: rule = (l,r) by force
  show ?case
  proof (cases l)
    case (Var v)

```

```

    with Cons Pair show ?thesis by simp
  next
    case (Fun f ts)
    with Cons Pair show ?thesis
    proof (cases computeRuleMapH rules)
      case (Some rrm) note oSome = this
      let ?dis = distinct-eq (λ (f,rls) (g,rls'). f = g)
      from Cons(1)[OF Some] have drrm: ?dis rrm and srrm: ?setl rrm = ?setr
rules by auto
      show ?thesis
      proof (cases List.extract (λ (fa,rls). fa = (f,length ts)) rrm)
        case None
        let ?e = ((f,length ts), [(Fun f ts,r)])
        let ?e' = ((f,length ts), {(Fun f ts,r)})
        from None Cons(2) have rm: rm = ?e # rrm by (simp add: Fun Pair
Some None)
        from None[unfolded extract-None-iff] have rrm:  $\bigwedge g \ n \ rl. ((g,n),rl) \in set$ 
 $rrm \implies (f,length ts) \neq (g,n)$  by auto
        then have rrm':  $\bigwedge g \ n \ rl. ((g,n),rl) \in ?setr \ rules \implies (f,length ts) \neq (g,n)$ 
by (simp only: srrm[symmetric], auto)
        then have id:  $\{(Fun f ts, r)\} = \{(l, ra). (l = Fun f ts \wedge ra = r \vee (l, ra) \in set \ rules) \wedge root \ l = Some \ (f, length \ ts)\}$  by force
        from rrm have dis: ?dis rm
        by (simp add: rm drrm, auto)
        have ?setl rm = insert ?e' (?setl rrm) by (simp add: rm)
        also have ... = insert ?e' (?setr rules) by (simp add: srrm)
        also have ... = ?setr ( (Fun f ts,r) # rules)
        proof (rule set-eqI, clarify)
          fix g n rls
          show  $((g,n),rls) \in insert \ ?e' \ (?setr \ rules) = (((g,n),rls) \in ?setr \ ((Fun \ f \ ts,r) \# \ rules))$ 
          proof (cases (g,n) = (f,length ts))
            case False
            then have  $((g,n),rls) \in insert \ ?e' \ (?setr \ rules) = (((g,n),rls) \in ?setr \ rules)$  by auto
            also have ... =  $((g,n),rls) \in ?setr \ ((Fun \ f \ ts,r) \# \ rules)$  using False
          by auto
        finally show ?thesis .
      next
        case True note oTrue = this
        show ?thesis
        proof (cases rls = {(Fun f ts, r)})
          case True
          with oTrue show ?thesis by (simp add: id, force)
        next
          case False
          show ?thesis using rrm[of g n rls] True False by (simp add: False
True id, auto)
        qed
    qed

```

```

      qed
    qed
  finally show ?thesis
    by (simp only: dis drmm, simp add: Pair Fun)
next
  case (Some res)
  obtain bef fg rls aft where res = (bef,(fg,rls),aft) by (cases res, force)
  from extract-SomeE[OF Some[simplified this]] Some[simplified this] have
rrm: rrm = bef @ ((f,length ts), rls) # aft
    and e: List.extract (λ (fa, rls). fa = (f,length ts)) rrm = Some (bef,
((f,length ts),rls),aft) by auto
    let ?e = ((f,length ts), (Fun f ts,r) # rls)
    let ?e' = ((f,length ts), insert (Fun f ts,r) (set rls))
    have ((f,length ts),set rls) ∈ ?setl rrm unfolding rrm by auto
    then have rls: set rls = {(l, r) | l r. (l, r) ∈ set rules ∧ root l = Some (f,
length ts)} using Cons(1)[OF oSome] by auto
    obtain ba where ba: ba = bef @ aft by auto
    from Cons(2) e ba have rm: rm = ?e # ba by (simp add: Fun Pair oSome
e)
    from drmm[simplified rrm] have dis: ?dis ba unfolding distinct-eq-append
ba by auto
    from drmm[simplified rrm] have dis: ?dis rm unfolding rm distinct-eq-append
ba by (auto simp: dis[simplified ba])
    from drmm[simplified rrm distinct-eq-append]
    have diff: (∀ x∈set ba. ¬ (λ(g, rls). (f,length ts) = g) x) by (auto simp: ba)
    have ?setl [((f, length ts),rls)] ∪ ?setl ba = ?setl rrm using rrm ba by auto
    also have ... = ?setr rules by (rule srrm)
    finally have id: ?setl [((f, length ts),rls)] ∪ ?setl ba = ?setr rules .
    have ?setl rm = insert ?e' (?setl ba) by (simp add: rm)
    also have ... = ?setr ((Fun f ts,r) # rules)
    proof (rule set-eqI, clarify)
      fix g n rl
      show (((g,n),rl) ∈ insert ?e' (?setl ba)) = (((g,n),rl) ∈ ?setr ((Fun f ts,r)
# rules))
    proof (cases (g,n) = (f,length ts))
      case False
      then have (((g,n),rl) ∈ insert ?e' (?setl ba)) = (((g,n),rl) ∈ ?setl ba)
by auto
      also have ... = (((g,n),rl) ∈ ?setr rules) using False by (simp only:
id[symmetric], auto)
      also have ... = (((g,n),rl) ∈ ?setr ((Fun f ts,r) # rules)) using False
by auto
      finally show ?thesis .
    next
    case True note oTrue = this
    show ?thesis
    proof (cases rl = insert (Fun f ts, r) (set rls))
      case True
      then have (((g,n),rl) ∈ insert ?e' (?setl ba)) = True using oTrue by

```

```

auto
  also have ... = (((g,n),rl) ∈ ?setr ((Fun f ts,r) # rules)) unfolding
True rls using oTrue by force
  finally show ?thesis .
next
  case False
  then have (((g,n),rl) ∈ insert ?e' (?setl ba)) = False using diff by
(simp add: True, auto)
  also have ... = (((g,n),rl) ∈ ?setr ((Fun f ts,r) # rules))
  proof (rule ccontr)
    assume ¬ ?thesis
    with True have ((f,length ts),rl) ∈ ?setr ((Fun f ts,r) # rules) by
simp
    then have rl = {(l, ra) | l ra. (l, ra) ∈ set ((Fun f ts, r) # rules) ∧
root l = Some (f, length ts)} by simp
    with False rls show False by auto
  qed
  finally show ?thesis .
qed
qed
qed
also have ... = ?setr (rule # rules) by (simp add: Pair Fun)
  finally show ?thesis by (simp add: dis)
qed
qed simp
qed
qed force

lemma computeRuleMapSound:
  assumes computeRuleMap R = Some rm
  shows (set (rm (f,n))) = {(l,r) | l r. (l,r) ∈ set R ∧ root l = Some (f, n)}
proof (cases computeRuleMapH R)
  case None
  then show ?thesis using assms unfolding computeRuleMap-def by auto
next
  case (Some rrm)
  note rrm = computeRuleMapHSound[OF this]
  note rm = assms[unfolded computeRuleMap-def, simplified Some, simplified,
symmetric]
  show ?thesis
  proof (cases map-of rrm (f, n))
    case (Some rls)
    from map-of-SomeD[OF this] have ((f,n),set rls) ∈ (λ (f,rls). (f, set rls)) ‘
set rrm
    by auto
    then have set rls = {(l,r) | l r. (l,r) ∈ set R ∧ root l = Some (f, n)}
    by (simp only: rrm, simp)
    then show ?thesis by (simp add: rm Some)
  next

```

```

case None
have id:  $\{(l, r) \mid l \text{ r. } (l, r) \in \text{set } R \wedge \text{root } l = \text{Some } (f, n)\} = \{\}$  (is ?set =  $\{\}$ )
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain l r where  $(l, r) \in \text{set } R \wedge \text{root } l = \text{Some } (f, n)$  by auto
  with rrm have  $((f, n), ?set) \in (\lambda (f, rls). (f, \text{set } rls)) \text{ 'set } rrm$  by auto
  with None[unfolded map-of-eq-None-iff] show False by force
qed
then show ?thesis by (simp only: rm None id, auto)
qed
qed

```

```

lemma computeRuleMap-left-vars:
  shows  $(\text{computeRuleMap } R \neq \text{None}) = (\forall lr \in \text{set } R. \forall x. \text{fst } lr \neq \text{Var } x)$ 
proof (cases computeRuleMap R)
  case None
    from None computeRuleMapSound2 have  $\exists (l, r) \in \text{set } R. \text{root } l = \text{None}$  by auto
    from this obtain l r where  $(l, r) \in \text{set } R \wedge \text{root } l = \text{None}$  by auto
    from this have  $(l, r) \in \text{set } R \wedge \neg (\forall x. \text{fst } (l, r) \neq \text{Var } x)$  by (cases l, auto)
    with None show ?thesis by blast
  next
    case (Some rm)
    then have left: computeRuleMap R ≠ None by auto
    from Some computeRuleMapSound2 have  $\forall (l, r) \in \text{set } R. \text{root } l \neq \text{None}$  by force
    then have  $\forall lr \in \text{set } R. \forall x. \text{fst } lr \neq \text{Var } x$  by auto
    with left show ?thesis by blast
qed

```

```

lemma computeRuleMap-defined: fixes R :: ('f, 'v)rules
  assumes computeRuleMap R = Some rm
  shows  $(\text{rm } (f, n) = []) = (\neg \text{defined } (\text{set } R) (f, n))$ 
proof –
  from assms computeRuleMapSound have rm:  $\bigwedge (f::'f) \ n. \text{set } (\text{rm } (f, n)) = \{(l, r) \mid l \text{ r. } (l, r) \in \text{set } R \wedge \text{root } l = \text{Some } (f, n)\}$  by force
  show ?thesis
  proof (cases rm (f, n))
    case Nil
    with rm have  $\neg \text{defined } (\text{set } R) (f, n)$  unfolding defined-def by force
    with Nil show ?thesis by blast
  next
    case (Cons lr RR)
    then have left: rm (f, n) ≠ [] by auto
    from Cons rm[where f = f and n = n] have defined (set R) (f, n) unfolding defined-def by (cases lr, force)
    with left show ?thesis by blast

```

qed
qed

lemma *computeRuleMap-None-not-SN*:
assumes *computeRuleMap R = None*
shows $\neg \text{SN-on } (\text{rstep } (\text{set } R)) \{t\}$
proof –
from *assms computeRuleMap-left-vars[of R]* **obtain** *x r* **where** $(\text{Var } x, r) \in \text{set } R$ **by** *auto*
from *left-var-imp-not-SN[OF this]* **show** *?thesis* .
qed

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]*: $\text{set } (\text{reverse-rules } R) = (\text{set } R)^{-1}$ **unfolding** *reverse-rules-def* **by** *force*

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*: $\text{set } (\text{map-funs-rules-wa fg } R) = \text{map-funs-trs-wa fg } (\text{set } R)$
unfolding *map-funs-rules-wa-def map-funs-trs-wa-def* **by** *auto*

lemma *wf-rule [code]*:
wf-rule r \longleftrightarrow
 $\text{is-Fun } (\text{fst } r) \wedge (\forall x \in \text{set } (\text{vars-term-impl } (\text{snd } r)). x \in \text{set } (\text{vars-term-impl } (\text{fst } r)))$
unfolding *wf-rule-def* **by** *auto*

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]*:
 $\text{set } (\text{wf-rules-impl } R) = \text{wf-rules } (\text{set } R)$
unfolding *wf-rules-impl-def wf-rules-def* **by** *auto*

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]:
  isOK (check-wf-reltrs (R, S)) = wf-reltrs (set R) (set S)
by (cases R) auto

declare check-wf-reltrs.simps[simp del]

definition check-linear-trs :: (·, ·) rules ⇒ showsl check where
  check-linear-trs R ≡
    check-all (λ (l,r). (linear-term l) ∧ (linear-term r)) R
    <+? (λ -. showsl-trs R ∘ showsl (STR '⌊↔⌋ is not linear ⌊↔⌋'))

lemma check-linear-trs [simp]:
  isOK (check-linear-trs R) ⌊↔⌋ linear-trs (set R)
unfolding check-linear-trs-def linear-trs-def by auto

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)
unfolding non-collapsing-impl-def non-collapsing-def list-all-iff by auto

type-synonym ('f, 'v) term-map = 'f × nat ⇒ ('f, 'v) term list

definition term-map :: ('f::compare-order, 'v) term list ⇒ ('f, 'v) term-map where
  term-map ts = fun-of-map (rm.α (elem-list-to-rm (the ∘ root) ts)) []

definition
  is-NF-main :: bool ⇒ bool ⇒ ('f::compare-order, 'v) term-map ⇒ ('f, 'v) term
  ⇒ bool
where
  is-NF-main var-cond R-empty m = (if var-cond then (λ-. False)
    else if R-empty then (λ-. True)
    else (λt. ∀ u∈set (supteq-list t).
      if is-Fun u then
        ∀ l∈set (m (the (root u))). ¬ matches u l
      else True))

lemma neq-root-no-match:
  assumes is-Fun l and the (root l) ≠ the (root t)
  shows ¬ matches t l
  using assms by (cases t) (force iff: matches-iff)+

lemma all-not-conv: (∀ x ∈ A. ¬ P x) = (¬ (∃ x ∈ A. P x)) by auto

lemma efficient-supteq-list-do-not-match:
  assumes ∀ l∈set ls. ∀ u∈set (supteq-list t). the (root l) ≠ the (root u) ⟶ ¬

```

$matches\ u\ l$
shows
 $(\forall l \in set\ ls. \forall u \in set\ (supteq-list\ t). \neg matches\ u\ l) \longleftrightarrow$
 $(\forall u \in set\ (supteq-list\ t). \forall l \in set\ (term-map\ ls\ (the\ (root\ u))).$
 $\neg matches\ u\ l)$
 $(is\ ?lhs \longleftrightarrow ?rhs\ is - \longleftrightarrow (\forall u \in set\ ?subs. \forall l \in set\ (?ls\ u). \neg matches\ u\ l))$
proof (intro iffI ballI)
fix $u\ l$ **assume** $\forall l \in set\ ls. \forall u \in set\ ?subs. \neg matches\ u\ l$ **and** $u \in set\ ?subs$
and $l \in set\ (?ls\ u)$
then show $\neg matches\ u\ l$
using $elem-list-to-rm.rm-set-lookup[of\ the\ \circ\ root\ ls\ the\ (root\ u)]$
by (auto simp: o-def term-map-def rm-set-lookup-def)
next
fix $l\ u$ **assume** $1: \forall u \in set\ ?subs. \forall l \in set\ (?ls\ u). \neg matches\ u\ l$
and $l \in set\ ls$ **and** $u \in set\ ?subs$
with $assms$ **have** $the\ (root\ l) \neq the\ (root\ u) \longrightarrow \neg matches\ u\ l$
and $\forall l \in set\ (?ls\ u). \neg matches\ u\ l$ **by** simp+
with $elem-list-to-rm.rm-set-lookup[of\ the\ \circ\ root\ ls\ the\ (root\ u)]$
and $\langle l \in set\ ls \rangle$
show $\neg matches\ u\ l$ **by** (auto simp: o-def term-map-def rm-set-lookup-def)
qed

lemma *supteq-list-ex*:
 $(\exists u \in set\ (supteq-list\ l). \exists \sigma. t \cdot \sigma = u) \longleftrightarrow (\exists \sigma. l \succeq t \cdot \sigma)$
unfolding *supteq-list* **by** auto

definition *is-NF-trs* $R = is-NF-main\ (\exists r \in set\ R. is-Var\ (fst\ r))\ (R = [])\ (term-map\ (map\ fst\ R))$
definition *is-NF-terms* $Q = is-NF-main\ (\exists q \in set\ Q. is-Var\ q)\ (Q = [])\ (term-map\ Q)$

lemma *is-NF-main-NF-trs-conv*:
 $is-NF-main\ (\exists r \in set\ R. is-Var\ (fst\ r))\ (R = [])\ (term-map\ (map\ fst\ R))\ t \longleftrightarrow$
 $t \in NF-trs\ (set\ R)$
 $(is\ is-NF-main\ ?var\ ?R\ ?map\ t \longleftrightarrow -)$
proof (intro iffI allI)
assume *is-NF-main*: $is-NF-main\ ?var\ ?R\ ?map\ t$
show $t \in NF-trs\ (set\ R)$
proof (cases $\exists r \in set\ R. is-Var\ (fst\ r)$)
case *True* **with** *is-NF-main*[*unfolded is-NF-main-def*] **show** *?thesis* **by** simp
next
case *False*
let $?ts = map\ fst\ R$
from *False* **have** *allfun*: $\forall s \in set\ ?ts. is-Fun\ s$ **by** simp
with *neg-root-no-match*
have *no-match*: $\forall s \in set\ ?ts. \forall u \in set\ (supteq-list\ t). the\ (root\ s) \neq the\ (root\ u)$
 $\longrightarrow \neg matches\ u\ s$ **by** blast
note *is-NF-main* = *is-NF-main*[*unfolded is-NF-main-def if-not-P[OF False]*]
show *?thesis*

```

proof (cases R = [])
  case False
  then have False: (R = []) = False by simp
  have  $\forall u \in \text{set } (\text{supteq-list } t). \forall l \in \text{set } (\text{term-map } ?ts (\text{the } (\text{root } u))). \neg \text{matches}$ 
u l
  proof
    fix u
    assume mem:  $u \in \text{set } (\text{supteq-list } t)$ 
    show  $\forall l \in \text{set } (\text{term-map } ?ts (\text{the } (\text{root } u))). \neg \text{matches } u \ l$ 
    proof (cases u)
      case (Var x)
      show ?thesis
    proof
      fix l
      assume  $l \in \text{set } (\text{term-map } ?ts (\text{the } (\text{root } u)))$ 
      with elem-list-to-rm.rm-set-lookup[of the  $\circ$  root ?ts the (root u)]
      have  $l \in \text{set } ?ts$  by (auto simp: o-def term-map-def rm-set-lookup-def)
      then have is-Fun l using allfun by auto
      then have  $(\forall \sigma. l \cdot \sigma \neq u)$  using Var by auto
      then show  $\neg \text{matches } u \ l$  using matches-iff by blast
    qed
  next
    case (Fun f us)
    with mem is-NF-main[unfolded False] show ?thesis by auto
  qed
qed
then show ?thesis
  unfolding efficient-supteq-list-do-not-match[OF no-match, symmetric]
  unfolding all-not-conv matches-iff
  unfolding supteq-list-ex by auto
qed auto
qed
next
  assume NF-trs:  $t \in \text{NF-trs } (\text{set } R)$ 
  show is-NF-main  $(\exists r \in \text{set } R. \text{is-Var } (\text{fst } r)) (R = []) (\text{term-map } (\text{map } \text{fst } R)) t$ 
  proof (cases  $\exists r \in \text{set } R. \text{is-Var } (\text{fst } r)$ )
    case True
    then obtain l where  $l \in \text{lhss } (\text{set } R)$  and is-Var l by auto
    from lhs-var-not-NF[OF this] and NF-trs show ?thesis by simp
  next
    case False note oFalse = this
    let ?ts = map fst R
    from False have  $\forall s \in \text{set } ?ts. \text{is-Fun } s$  by auto
    with neg-root-no-match
    have
      no-match:  $\forall s \in \text{set } ?ts. \forall u \in \text{set } (\text{supteq-list } t). \text{the } (\text{root } s) \neq \text{the } (\text{root } u)$ 
       $\longrightarrow \neg \text{matches } u \ s$  by blast
    show ?thesis
  proof (cases R = [])

```

```

    case True then show ?thesis unfolding is-NF-main-def by auto
next
case False
then have False: (R = []) = False by simp
from NF-trs
show ?thesis
  unfolding is-NF-main-def False if-False if-not-P[OF oFalse]
  using efficient-supteq-list-do-not-match[OF no-match, symmetric]
  unfolding all-not-conv matches-iff
  unfolding supteq-list-ex set-map by fastforce
qed
qed
qed

```

```

lemma is-NF-trs [simp]:
  is-NF-trs R = (λ t. t ∈ NF-trs (set R))
  by (rule ext, unfold is-NF-trs-def is-NF-main-NF-trs-conv, simp)

```

```

lemma is-NF-terms [simp]:
  is-NF-terms Q = (λ t. t ∈ NF-terms (set Q))
proof (rule ext)
  fix t
  let ?Q = map (λt. (t, t)) Q
  have 1: (∃ t ∈ set Q. is-Var t) = (∃ t ∈ set ?Q. is-Var (fst t))
    by (induct Q) (auto simp: o-def)
  have 2: map fst ?Q = Q by (induct Q) simp-all
  have 3: term-map Q = term-map (map fst ?Q) unfolding 2 ..
  have 4: set ?Q = Id-on (set Q) by (induct Q) (auto simp: o-def)
  from is-NF-main-NF-trs-conv[of ?Q t]
  show is-NF-terms Q t = (t ∈ NF-terms (set Q))
    unfolding is-NF-terms-def 1 3 4 unfolding 2 by auto
qed

```

```

abbreviation terms-of-rules rs ≡ map fst rs @ map snd rs

```

```

definition match-rules rs1 rs2 = do {
  ts ← zip-option (terms-of-rules rs2) (terms-of-rules rs1);
  match-list Var ts
}

```

```

abbreviation term-of-rules rs ≡ Fun (SOME f. True) (terms-of-rules rs)

```

```

lemma match-rules-alt-def:
  match-rules rs1 rs2 = match (term-of-rules rs1) (term-of-rules rs2)
  by (cases zip-option (terms-of-rules rs1) (terms-of-rules rs2))
    (auto simp: match-rules-def match-def match-list-def decompose-def
      split: option.splits Option.bind-splits)

```

```

lemma rules-enc-aux:

```

```

fixes  $xs\ ys :: ('f, 'v :: infinite)\ rules$ 
assumes  $map ((\cdot)\ \pi \circ fst)\ xs\ @\ map ((\cdot)\ \pi \circ snd)\ xs = map\ fst\ ys\ @\ map\ snd\ ys$ 
shows  $map ((\cdot)\ \pi)\ xs = ys$ 
proof -
  have  $length\ xs = length\ ys$ 
  proof (rule ccontr)
    assume  $length\ xs \neq length\ ys$ 
    moreover
      have  $length\ (map ((\cdot)\ \pi \circ fst)\ xs\ @\ map ((\cdot)\ \pi \circ snd)\ xs) = length\ (map\ fst\ ys\ @\ map\ snd\ ys)$ 
    using assms by simp
    ultimately show False by simp
  qed
  then show ?thesis
    using assms by (induct  $xs\ ys$  rule: list-induct2; auto simp: eqvt)
qed

```

```

lemma perm-term-of-rules-eq-conv:
   $\pi \cdot term-of-rules\ rs_1 = term-of-rules\ rs_2 \longleftrightarrow \pi \cdot rs_1 = rs_2$ 
by (auto simp: eqvt rules-enc-aux)

```

```

lemma match-rules-imp-variants:
  fixes  $rs_1\ rs_2 :: ('f, 'v :: infinite)\ rules$ 
  assumes  $match-rules\ rs_2\ rs_1 = Some\ \sigma_1$  and  $match-rules\ rs_1\ rs_2 = Some\ \sigma_2$ 
  shows  $\exists \pi :: 'v\ perm.\ \pi \cdot rs_1 = rs_2$ 
proof -
  have  $term-of-rules\ rs_2 \cdot \sigma_2 = term-of-rules\ rs_1$ 
  and  $term-of-rules\ rs_1 \cdot \sigma_1 = term-of-rules\ rs_2$ 
  using assms by (auto simp: match-rules-alt-def dest: match-sound)
  then obtain  $\pi :: 'v\ perm$  where  $\pi \cdot rs_2 = rs_1$ 
  using term-variants-iff [of  $term-of-rules\ rs_2\ term-of-rules\ rs_1$ ]
  unfolding perm-term-of-rules-eq-conv by auto
  then have  $-\pi \cdot rs_1 = rs_2$  by (auto simp del: rules-pt.permute-list-def)
  then show ?thesis ..
qed

```

```

definition
  check-variants-rule  $r\ r' = do\ \{$ 
     $check\ (match-rules\ [r]\ [r'] \neq None \wedge match-rules\ [r']\ [r] \neq None)$ 
     $(showsl-rule\ r\ \circ\ showsl\ (STR\ ''\ and\ '')\ \circ\ showsl-rule\ r'\ \circ\ showsl\ (STR\ ''\ are$ 
     $not\ variants\ of\ each\ other\ [\longleftrightarrow]''))$ 
   $\}$ 

```

```

lemma check-variants-rule [elim!]:
  assumes isOK (check-variants-rule  $r\ r'$ )
  obtains  $p$  where  $p \cdot r = r'$ 
  using assms
  by (cases  $r$ ; cases  $r'$ ) (auto simp: check-variants-rule-def eqvt dest: match-rules-imp-variants)

```

definition

check-variant-in-trs R $r =$
check-exm (*check-variants-rule* r) R (*showsl-sep id id*)
 $<+?$ ($\lambda-. \text{showsl-rule } r \circ \text{showsl } (STR \text{ '' is not a variant of any rule in ''}) \circ$
showsl-trs R)

lemma *check-variant-in-trs* [elim!]:

assumes *isOK* (*check-variant-in-trs* R r)
obtains p **where** $p \cdot r \in \text{set } R$
using *assms* **by** (*cases* r) (*force simp: check-variant-in-trs-def eqvt*)

definition *check-variants-trs* R $R' = \text{check-allm } (\text{check-variant-in-trs } R') R$

lemma *check-variants-trs* [dest!]:

assumes *isOK* (*check-variants-trs* R R')
shows $\forall r \in \text{set } R. \exists p. p \cdot r \in \text{set } R'$
using *assms* **by** (*auto simp: check-variants-trs-def*)

lemma *check-variants-trs-rstep*: **assumes** *isOK* (*check-variants-trs*

$(R :: (f :: \text{showl}, 'v :: \{\text{infinite}, \text{showl}\}) \text{rules}) S$)

shows *rstep* (*set* R) \subseteq *rstep* (*set* S)

using *check-variants-trs* [OF *assms*]

by (*metis* (*no-types*, *opaque-lifting*) *perm-rstep-conv* *rstep-mono* *rstep-simps*(2)
subset-code(1) *subset-rstep*)

end

theory *Rewrite-Relations-Impl*

imports

Trs-Impl

Par-Step-Var-Restricted

Multistep

begin

16.2 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

is-par-rstep-var-restr (*Fun* f ss) (*Fun* g ts) =

$(\text{Fun } f \text{ } ss = \text{Fun } g \text{ } ts \vee$

$\text{vars-term } (\text{Fun } g \text{ } ts) \cap V = \{\} \wedge (\text{Fun } f \text{ } ss, \text{Fun } g \text{ } ts) \in \text{rrstep } (\text{set } R) \vee$

$(f = g \wedge \text{length } ss = \text{length } ts \wedge \text{list-all2 } \text{is-par-rstep-var-restr } ss \text{ } ts))$

| *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*: $\text{vars-term } t \cap V = \{\} \longleftrightarrow$
 $(\forall x \in \text{set } (\text{vars-term-list } t). x \notin V)$
by *auto*

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 $s\ t \longleftrightarrow (s, t) \in \text{par-rstep-var-restr } (\text{set } R)\ V$

proof

let $?Prop = \lambda s\ t. s = t \vee \text{vars-term } t \cap V = \{\} \wedge (s, t) \in \text{rrstep } (\text{set } R)$

{

fix $s\ t$

assume $?Prop\ s\ t$

hence $\exists\ C\ \text{infos. } (s, t) \in \text{par-rstep-mctxt } (\text{set } R)\ C\ \text{infos} \wedge \text{vars-below-hole } t$

$C \cap V = \{\}$

proof

assume $s = t$

thus $?thesis$ **by** (*intro exI*[*of - mctxt-of-term s*] *exI*[*of - Nil*], *auto simp: par-rstep-mctxt-refl*)

next

assume $\text{vars-term } t \cap V = \{\} \wedge (s, t) \in \text{rrstep } (\text{set } R)$

then obtain $l\ r\ \sigma$ **where** $id: s = l \cdot \sigma\ t = r \cdot \sigma$ **and**

$lr: (l, r) \in \text{set } R$ **and**

$\text{vars: vars-term } t \cap V = \{\}$

by (*metis rrstepE*)

thus $?thesis$ **by** (*intro exI*[*of - MHole*] *exI*[*of - [Par-Info s t (l,r)]*], *auto intro: par-rstep-mctxt-MHoleI*)

qed

} **note** $Prop = this$

{

assume *is-par-rstep-var-restr* $s\ t$

hence $\exists\ C\ \text{infos. } (s, t) \in \text{par-rstep-mctxt } (\text{set } R)\ C\ \text{infos} \wedge \text{vars-below-hole } t$

$C \cap V = \{\}$

proof (*induct rule: is-par-rstep-var-restr.induct*[])

case *2-1*

thus $?case$ **by** (*intro Prop, auto*)

next

case *2-2*

thus $?case$ **by** (*intro Prop, auto*)

next

case (*1 f ss g ts*)

show $?case$

proof (*cases ?Prop (Fun f ss) (Fun g ts)*)

case *True*

thus $?thesis$ **using** $Prop$ **by** *auto*

next

```

    case False
  with 1 have args:  $f = g$  length  $ss = \text{length } ts$  list-all2 is-par-rstep-var-restr
    ss ts
    by (auto split: if-splits)
    let  $?P = \lambda i \ C \ \text{infos}. (ss ! i, ts ! i) \in \text{par-rstep-mctxt} (\text{set } R) \ C \ \text{infos} \wedge$ 
    vars-below-hole  $(ts ! i) \ C \subseteq (UNIV - V)$ 
    { fix i
      assume  $i : i < \text{length } ss$ 
      then have  $si : ss ! i \in \text{set } ss$  by auto
      from i args(2) have  $ti : ts ! i \in \text{set } ts$  by auto
      from args(3) have  $iprv : \text{is-par-rstep-var-restr } (ss ! i) (ts ! i)$  using i
    list-all2-nthD by blast
      with 1(1)[of  $ss ! i \ ts ! i$ ] have  $pp : \exists C \ \text{infos}. ?P \ i \ C \ \text{infos}$ 
      using local.args(1) local.args(2) using si ti by blast
    }
    hence  $\forall i. \exists C \ \text{infos}. i < \text{length } ss \longrightarrow ?P \ i \ C \ \text{infos}$  by blast
    from choice[OF this] obtain C where  $\forall i. \exists \ \text{infos}. i < \text{length } ss \longrightarrow ?P$ 
    i (C i) infos by blast
    from choice[OF this] obtain infos where IH:  $\bigwedge i. i < \text{length } ss \implies ?P \ i$ 
    (C i) (infos i) by blast
    let  $?C = \text{MFun } f \ (\text{map } C \ [0..<\text{length } ss])$ 
    let  $?infos = \text{concat } (\text{map } \text{infos} \ [0..<\text{length } ss])$ 
    show ?thesis
    proof (intro exI[of - ?C] exI[of - ?infos] conjI)
      show vars-below-hole  $(\text{Fun } g \ ts) \ ?C \cap V = \{\}$  using IH args(2) unfolding
    args(1)
      by (subst vars-below-hole-Fun; force)
      show  $(\text{Fun } f \ ss, \text{Fun } g \ ts) \in \text{par-rstep-mctxt} (\text{set } R) \ ?C \ ?infos$  unfolding
    args(1) using args(2) IH
      by (intro par-rstep-mctxt-funI, auto)
    qed
  qed
  qed
  thus  $(s, t) \in \text{par-rstep-var-restr} (\text{set } R) \ V$  unfolding par-rstep-var-restr-def
  by auto
}
{
  assume  $(s, t) \in \text{par-rstep-var-restr} (\text{set } R) \ V$ 
  from this[unfolded par-rstep-var-restr-def] obtain C infos where
    st:  $(s, t) \in \text{par-rstep-mctxt} (\text{set } R) \ C \ \text{infos}$  and vars: vars-below-hole  $t \ C \cap$ 
  V =  $\{\}$  by auto
  thus is-par-rstep-var-restr s t
  proof (induct C arbitrary: s t infos)
    case (MVar x)
    from par-rstep-mctxt-MVarE[OF MVar(1)]
    have  $s = \text{Var } x \ t = \text{Var } x$  by auto
    thus ?case by simp
  next
    case MHole

```

```

from par-rstep-mctxt-MHoleE[OF MHole(1)]
have  $(s, t) \in \text{rrstep } (\text{set } R)$  by auto
then show  $?case$  using MHole(2) by (cases s; cases t; auto)
next
case (MFun f Cs)
from par-rstep-mctxt-MFunD[OF MFun(2)]
obtain ss ts Infos
where  $s = \text{Fun } f \text{ } ss$  and
 $t = \text{Fun } f \text{ } ts$  and
 $\text{length } ss = \text{length } Cs$ 
 $\text{length } ts = \text{length } Cs$ 
 $\text{length } Infos = \text{length } Cs$  and
 $infos = \text{concat } Infos$  and
 $\text{steps: } \bigwedge i. i < \text{length } Cs \implies (ss ! i, ts ! i) \in \text{par-rstep-mctxt } (\text{set } R) (Cs !$ 
i) (Infos ! i)
by auto
show  $?case$  unfolding  $s \text{ } t$  is-par-rstep-var-restr.simps
proof (intro disjI2 conjI refl list-all2-all-nthI, unfold len)
fix i
assume  $i < \text{length } Cs$ 
hence  $mem: Cs ! i \in \text{set } Cs$  by auto
show  $\text{is-par-rstep-var-restr } (ss ! i) (ts ! i)$ 
proof (rule MFun(1)[OF mem steps[OF i]])
have  $\text{vars-below-hole } (ts ! i) (Cs ! i) \subseteq \text{vars-below-hole } t (MFun f Cs)$ 
unfolding  $t$  using  $i \text{ } len$ 
by (subst vars-below-hole-Fun, auto)
with MFun(3) show  $\text{vars-below-hole } (ts ! i) (Cs ! i) \cap V = \{\}$  by auto
qed
qed auto
qed
qed
qed
qed
qed

```

```

lemma par-rstep-var-restr-code[code-unfold]:
 $(s, t) \in \text{par-rstep-var-restr } (\text{set } R) \text{ } V \longleftrightarrow \text{is-par-rstep-var-restr } R \text{ } V \text{ } s \text{ } t$ 
by simp

```

16.3 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
 $\text{is-par-rstep } R (Fun f ss) (Fun g ts) =$ 
 $(Fun f ss = Fun g ts \vee (Fun f ss, Fun g ts) \in \text{rrstep } (\text{set } R) \vee$ 
 $(f = g \wedge \text{length } ss = \text{length } ts \wedge \text{list-all2 } (\text{is-par-rstep } R) \text{ } ss \text{ } ts))$ 
 $| \text{is-par-rstep } R \text{ } s \text{ } t = (s = t \vee (s, t) \in \text{rrstep } (\text{set } R))$ 

```

```

lemma is-par-rstep[simp]:
 $\text{is-par-rstep } R \text{ } s \text{ } t \longleftrightarrow (s, t) \in \text{par-rstep } (\text{set } R)$ 

```

proof –

have *is-par-rstep* R s $t = is-par-rstep-var-restr$ R $\{ \}$ s t
 by (*induct* R s t *rule: is-par-rstep.induct*, *auto simp del: is-par-rstep-var-restr*
simp: list-all2-conv-all-nth)
 also have $\dots \longleftrightarrow (s, t) \in par-rstep-var-restr$ (*set* R) $\{ \}$ **by** *simp*
 also have $\dots \longleftrightarrow (s, t) \in par-rstep$ (*set* R)
 unfolding *par-rstep-var-restr-def par-rstep-par-rstep-mctxt-conv* **by** *auto*
 finally show *?thesis* .
qed

lemma *par-rstep-code[code-unfold]*: $(s, t) \in par-rstep$ (*set* R) $\longleftrightarrow is-par-rstep$ R s t **by** *simp*

16.4 Generate all root-rewrites (for well-formed TRS)

fun *root-rewrite* :: $(f, 'v)$ *rules* $\Rightarrow (f, 'v)$ *term* $\Rightarrow (f, 'v)$ *term list*

where

root-rewrite R $s = concat$ (*map* $(\lambda (l, r).$
 (*case match* s l *of*
 $None \Rightarrow []$
 $| Some \sigma \Rightarrow [(r \cdot \sigma)])$ R)

lemma *root-rewrite-sound*:

assumes $t \in set$ (*root-rewrite* R s)
 shows $(s, t) \in rstep$ (*set* R)

proof –

from *assms*
 have $\exists l r. (l, r) \in set$ $R \wedge t \in set$ (*case match* s l *of* $None \Rightarrow [] \mid Some \sigma \Rightarrow [r \cdot \sigma]$)
 by *auto*
 from this obtain $l r$ **where** *one*:
 $(l, r) \in set$ $R \wedge t \in set$ (*case match* s l *of* $None \Rightarrow [] \mid Some \sigma \Rightarrow [r \cdot \sigma]$)
 by *auto*
 from this obtain σ **where** *two*: *match* s $l = Some \sigma \wedge t \in \{r \cdot \sigma\}$ **by** (*cases match* s l , *auto*)
 then have *match*: $l \cdot \sigma = s$ **using** *match-sound* **by** *auto*
 with one match one two have $(s, t) \in rstep-r-p-s$ (*set* R) (l, r) $\square \sigma$ **unfolding**
rstep-r-p-s-def **by** (*simp add: Let-def ctxt-supt-id*)
 then show $(s, t) \in rstep$ (*set* R) **unfolding** *rstep-iff-rstep-r-p-s rstep-def* **by**
blast
qed

16.5 Generate all parallel rewrites (for well formed TRS)

fun *parallel-rewrite* :: $(f, 'v)$ *rules* $\Rightarrow (f, 'v)$ *term* $\Rightarrow (f, 'v)$ *term list*

where

parallel-rewrite R (*Var* x) = [*Var* x]
 $| parallel-rewrite$ R (*Fun* f ss) = *remdups*
 (*root-rewrite* R (*Fun* f ss) @ *map* $(\lambda ss. Fun$ f $ss)$ (*product-lists* (*map* (*parallel-rewrite*
 R) ss)))

```

lemma parallel-rewrite-par-step:
  assumes  $t \in \text{set } (\text{parallel-rewrite } R \ s)$ 
  shows  $(s, t) \in \text{par-rstep } (\text{set } R)$ 
using assms
proof (induct s arbitrary: t)
  case (Fun f ss)
  then consider (root)  $t \in \text{set } (\text{root-rewrite } R \ (\text{Fun } f \ ss))$ 
    | (args)  $t \in \text{set } (\text{map } (\lambda ss. \text{Fun } f \ ss) \ (\text{product-lists } (\text{map } (\text{parallel-rewrite } R) \ ss)))$ 
  by force
  then show ?case
  proof (cases)
  case root
  from root-rewrite-sound[OF this] obtain  $l \ r \ \sigma$  where  $(l, r) \in \text{set } R$ 
    and  $l \cdot \sigma = \text{Fun } f \ ss$  and  $r \cdot \sigma = t$ 
  unfolding rrstep-def rstep-r-p-s-def by auto
  then show ?thesis using par-rstep.intros(1) by metis
next
  case args
  then obtain  $ts$  where  $t:t = \text{Fun } f \ ts$  and  $ts:ts \in \text{set } (\text{product-lists } (\text{map } (\text{parallel-rewrite } R) \ ss))$ 
  by auto
  then have  $\text{len:length } ss = \text{length } ts$  using in-set-product-lists-length by force
  { fix  $i$ 
    assume  $i:i < \text{length } ts$ 
    have  $ts ! i \in \text{set } (\text{parallel-rewrite } R \ (ss ! i))$ 
    using  $ts[\text{unfolded product-lists-set}[of \ - \ ss]]$ 
    by (auto simp: list-all2-map2[ $of \ (\lambda x \ ys. x \in \text{set } ys)$ ] intro: list-all2-nthD[OF -  $i$ ])
    with  $\text{Fun}(1) \ \text{len } i$  have  $(ss ! i, ts ! i) \in \text{par-rstep } (\text{set } R)$  by auto
  }
  from par-rstep.intros(2)[OF this len] show ?thesis using  $t$  by auto
qed
qed auto

```

```

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
   $\text{root-steps-substs } R \ s \ t = \text{concat } (\text{map } (\lambda (l, r). \text{case match } s \ l \text{ of}$ 
     $\text{None} \Rightarrow []$ 
    |  $\text{Some } \sigma \Rightarrow (\text{case match } t \ r \text{ of}$ 
       $\text{None} \Rightarrow []$ 
      |  $\text{Some } \tau \Rightarrow (\text{let var-list} = \text{filter } (\lambda x. x \in \text{vars-term } r) \ (\text{vars-distinct } l) \text{ in}$ 
         $[(\text{map } \sigma \ \text{var-list}, \text{map } \tau \ \text{var-list})])))$ 
     $R)$ 

```

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–

from *assms* **obtain** $l \ r$ **where** $lr:(l,r) \in \text{set } R \ (ss, ts) \in \text{set } (\text{case match } s \ l \ \text{of}$
 $\text{None} \Rightarrow []$
 $| \text{Some } \sigma \Rightarrow (\text{case match } t \ r \ \text{of}$
 $\text{None} \Rightarrow []$
 $| \text{Some } \tau \Rightarrow [(\text{map } \sigma (\text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l)), \text{map } \tau$
 $(\text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l)))]])$
unfolding *root-steps-substs.simps Let-def* **by** *auto*
let $?var\text{-list} = \text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l)$
from lr **obtain** σ **where** $\sigma:\text{match } s \ l = \text{Some } \sigma$
by *fastforce*
from lr **obtain** τ **where** $\tau:\text{match } t \ r = \text{Some } \tau$
by *fastforce*
from $lr \ \sigma \ \tau$ **have** $(ss, ts) = (\text{map } \sigma \ ?var\text{-list}, \text{map } \tau \ ?var\text{-list})$
by *simp*
with $lr(1) \ \sigma \ \tau$ **show** *?thesis*
using *match-matches* **by** *blast*
qed

lemma *size-match-subst-Fun*:

assumes *is-Fun* l **and** $x \in \text{vars-term } l$
and $\text{match}:\text{match } s \ l = \text{Some } \tau$
shows $\text{size } (\tau \ x) < \text{size } s$
proof–
from *assms*(1) **obtain** $f \ ts$ **where** $l:l = \text{Fun } f \ ts$
by *blast*
from *match* **have** $*:l \cdot \tau = s$
by (*simp add: match-matches*)
then obtain ss **where** $s:s = \text{Fun } f \ ss$
unfolding l **by** *force*
from *assms*(2) **obtain** i **where** $i:i < \text{length } ts$ **and** $x:x \in \text{vars-term } (ts!i)$
unfolding l **by** (*metis term.sel(4) var-imp-var-of-arg*)
from $*$ **have** $le:\text{length } ts = \text{length } ss$
unfolding $s \ l$ **by** *auto*
moreover from $* \ i \ l \ s$ **have** $ts!i \cdot \tau = ss!i$
by *fastforce*
then have $\text{size } (\tau \ x) \leq \text{size } (ss!i)$
using *vars-term-size x* **by** *metis*
with i **show** *?thesis* **unfolding** $s \ \text{term.size } le$
by (*metis add.commute add-0 add-Suc in-set-conv-nth less-Suc-eq-le size-list-estimation'*)
qed

16.6 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–

from *assms* **obtain** $x y$ **where** $xy: (\text{Var } x, \text{Var } y) \in R \wedge x \neq y$ **by** *blast*

let $?s = (\text{subst } x s) (y := t)$

from xy **have** $(?s x, ?s y) \in \text{rrstep } R$

by $(\text{metis eval-term.simps}(1) \text{ rrstepI})$

then show *?thesis*

by $(\text{simp add: } xy(2))$

qed

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–

let $?R = \text{remove-trivial-rules } R$

from *assms*(1) **obtain** $l r vl \sigma \tau$ **where** $lr: (l, r) \in \text{set } ?R$ **and** $vl: vl = \text{filter } (\lambda x. x \in \text{vars-term } r) (\text{vars-distinct } l)$

and $s:s = l \cdot \sigma$ **and** $t:t = r \cdot \tau$ **and** $ss-ts: (ss, ts) = (\text{map } \sigma vl, \text{map } \tau vl)$

using *root-steps-substs-exists* **by** *blast*

from $ss-ts$ *assms*(2) **obtain** x **where** $s':s' = \sigma x$ **and** $x:x \in \text{set } vl$

by *auto*

with s **have** $s1:\text{size } s' \leq \text{size } s$

unfolding vl **by** $(\text{simp add: vars-term-size})$

from $ss-ts$ *assms*(3) **obtain** y **where** $t':t' = \tau y$ **and** $y:y \in \text{set } vl$

by *auto*

with t **have** $s2:\text{size } t' \leq \text{size } t$

unfolding vl **by** $(\text{simp add: vars-term-size})$

from lr **consider** $\neg \text{is-Var } l \mid \neg \text{is-Var } r$

by *force*

then show *?thesis* **proof**(*cases*)

case 1

then obtain $f ls$ **where** $l:l = \text{Fun } f ls$

by *blast*

from x **obtain** i **where** $i < \text{length } ls$ **and** $x \in \text{vars-term } (ls[i])$

unfolding $vl l$ **by** $(\text{metis comp-apply filter-is-subset set-remdups set-rev set-vars-term-list subsetD term.sel}(4) \text{ var-imp-var-of-arg})$

then have $s' \triangleleft s$

unfolding $s s' l$ **by** $(\text{meson nth-mem subst-image-subterm term.set-intros}(4))$

then have $\text{size } s' < \text{size } s$

by $(\text{simp add: supt-size})$

then show *?thesis* **using** $s2$ **by** *simp*

next

```

case 2
  then obtain  $f\ rs$  where  $r:r = \text{Fun } f\ rs$ 
  by blast
  from  $y$  obtain  $i$  where  $i < \text{length } rs$  and  $y \in \text{vars-term } (rs!i)$ 
  unfolding  $vl\ r$  using var-imp-var-of-arg by force
  then have  $t' \triangleleft t$ 
  unfolding  $t\ t'\ r$  by (meson nth-mem subst-image-subterm term.set-intros(4))
  then have  $\text{size } t' < \text{size } t$ 
  by (simp add: supt-size)
  then show ?thesis using  $s1$  by simp
qed
qed

function (sequential) is-mstep ::  $(f, 'v)$  rules  $\Rightarrow (f, 'v)$  term  $\Rightarrow (f, 'v)$  term  $\Rightarrow$ 
bool
where
  is-mstep  $R\ (\text{Fun } f\ ss)\ (\text{Fun } g\ ts) =$ 
     $(\text{Fun } f\ ss = \text{Fun } g\ ts \vee (\text{Fun } f\ ss, \text{Fun } g\ ts) \in \text{rrstep } (\text{set } R) \vee$ 
    list-ex  $(\lambda (ss, ts). \text{list-all2 } (\text{is-mstep } R)\ ss\ ts) (\text{root-steps-substs } (\text{remove-trivial-rules}$ 
     $R)\ (\text{Fun } f\ ss)\ (\text{Fun } g\ ts)) \vee$ 
     $(f = g \wedge \text{length } ss = \text{length } ts \wedge \text{list-all2 } (\text{is-mstep } R)\ ss\ ts))$ 
  | is-mstep  $R\ s\ t = (s = t \vee (s, t) \in \text{rrstep } (\text{set } R) \vee$ 
    list-ex  $(\lambda (ss, ts). \text{list-all2 } (\text{is-mstep } R)\ ss\ ts) (\text{root-steps-substs } (\text{remove-trivial-rules}$ 
     $R)\ s\ t))$ 
  by pat-completeness auto

termination proof (relation measure  $(\lambda (R, s, t). \text{size } s + \text{size } t)$ , goal-cases)
  case (2  $R\ f\ ss\ g\ ts\ x\ ls\ rs\ l\ r$ )
  then show ?case using size-root-steps-substs
  unfolding in-measure by (metis case-prod-conv)
next
  case (3  $R\ f\ ss\ g\ ts\ s\ t$ )
  then have  $\text{size } s < \text{size } (\text{Fun } f\ ss)$  and  $\text{size } t < \text{size } (\text{Fun } g\ ts)$ 
  by (simp add: elem-size-size-list-size less-Suc-eq)+
  then show ?case by simp
next
  case (4  $R\ v\ t\ x\ xa\ y\ z\ yb$ )
  then show ?case using size-root-steps-substs
  unfolding in-measure by (metis case-prod-conv)
next
  case (5  $R\ s\ v\ x\ xa\ y\ z\ yb$ )
  then show ?case using size-root-steps-substs
  unfolding in-measure by (metis case-prod-conv)
qed auto

```

Show that all multisteps are covered by the definition above.

lemma *mstep-is-mstep*:
assumes $(s, t) \in \text{mstep } (\text{set } R)$
shows *is-mstep* $R\ s\ t$

```

using assms proof(induct)
case (args f n ss ts)
then have list-all2 (is-mstep R) ss ts
  by (simp add: list-all2-all-nthI)
with args show ?case
  by simp
next
case (rule l r  $\sigma$   $\tau$ )
show ?case proof(cases ( $l \cdot \sigma, r \cdot \tau$ )  $\in$  rrstep (set R))
  case True
  then show ?thesis using is-mstep.simps by (metis (no-types, opaque-lifting)
funas-term.cases)
next
case False
then show ?thesis proof(cases is-Var l  $\wedge$  is-Var r)
  case True
  with False have  $\neg (\exists x y. (Var\ x, Var\ y) \in set\ R \wedge x \neq y)$ 
    using trivial-rrstep by metis
  with True obtain x where  $l:l = Var\ x$  and  $r:r = Var\ x$ 
    using rule.hyps(1) by blast
  show ?thesis
    unfolding l r using rule(2) l by simp
next
case False
let ?R=remove-trivial-rules R
let ?vl=filter ( $\lambda x. x \in vars-term\ r$ ) (vars-distinct l)
from rule(1) False obtain i where  $i:i < length\ ?R$  ?R!i = (l, r)
by (metis (no-types, lifting) case-prodI2 fst-conv in-set-conv-nth mem-Collect-eq
prod.sel(2) set-filter)
obtain  $\sigma'$  where sigma:match ( $l \cdot \sigma$ ) l = Some  $\sigma'$  ( $\forall x \in vars-term\ l. \sigma\ x =$ 
 $\sigma'\ x$ )
  by (meson match-complete')
obtain  $\tau'$  where tau:match ( $r \cdot \tau$ ) r = Some  $\tau'$  ( $\forall x \in vars-term\ r. \tau\ x = \tau'$ 
 $x$ )
  by (meson match-complete')
let ?matches=(map ( $\lambda(l', r'). case$ 
  match ( $l \cdot \sigma$ ) l' of None  $\Rightarrow []$  | Some  $\sigma \Rightarrow (case\ match\ (r \cdot \tau)\ r'\ of\ None$ 
 $\Rightarrow []$ 
  | Some  $\tau \Rightarrow (let\ var-list = filter\ (\lambda x. x \in vars-term\ r')\ (vars-distinct\ l')$ 
  in [(map  $\sigma\ var-list, map\ \tau\ var-list$ )]))) ?R)
have  $i < length\ ?matches$ 
  using i(1) by auto
moreover have (map  $\sigma'\ ?vl, map\ \tau'\ ?vl$ )  $\in set\ (?matches\ !\ i)$ 
  using sigma(1) tau(1) i unfolding Let-def by simp
ultimately have (map  $\sigma'\ ?vl, map\ \tau'\ ?vl$ )  $\in set\ (root-steps-substs\ ?R\ (l \cdot \sigma)$ 
 $(r \cdot \tau))$ 
  unfolding root-steps-substs.simps by (metis (no-types, lifting) concat-nth
concat-nth-length in-set-conv-nth)
  then obtain j where  $j:j < length\ (root-steps-substs\ ?R\ (l \cdot \sigma)\ (r \cdot \tau))$  root-steps-substs

```

```

?R (l.σ) (r.τ) ! j = (map σ' ?vl, map τ' ?vl)
  by (metis in-set-idx)
  have (λ (ss, ts). list-all2 (is-mstep R) ss ts) ((root-steps-substs ?R (l.σ)
(r.τ))!j)
  proof-
    {fix i assume i:i < length ?vl
      from i have vr:?vl!i ∈ vars-term r
        using nth-mem by force
      from i have vl:?vl!i ∈ vars-term l
        using nth-mem by force
      moreover have σ' (?vl ! i) = σ (?vl ! i)
        using sigma(2) vr vl by simp
      moreover have τ' (?vl ! i) = τ (?vl ! i)
        using vl vr tau(2) by simp
      ultimately have is-mstep R (σ' (?vl ! i)) (τ' (?vl ! i))
        using rule(2) by force
    }
  then have list-all2 (is-mstep R) (map σ' ?vl) (map τ' ?vl)
    by (simp add: list-all2-conv-all-nth)
  then show ?thesis
    unfolding j(2) by fastforce
qed
then have *:list-ex (λ (ss, ts). list-all2 (is-mstep R) ss ts) (root-steps-substs
?R (l.σ) (r.τ))
  using j by (meson list-ex-length)
then show ?thesis
  by (smt (verit) is-mstep.elims(3))
qed
qed
qed simp

```

lemma *mstep-root-helper*:

```

  assumes list-ex (λ (ss, ts). list-all2 (is-mstep R) ss ts) (root-steps-substs (remove-trivial-rules
R) s t)
    and ∧ ss ts s' t'. (ss, ts) ∈ set (root-steps-substs (remove-trivial-rules R) s t)
    ⇒ s' ∈ set ss ⇒ t' ∈ set ts ⇒ is-mstep R s' t' ⇒ (s', t') ∈ mstep (set R)
  shows (s, t) ∈ mstep (set R)
  proof-
    let ?R=(remove-trivial-rules R)
    from assms obtain i where i < length (root-steps-substs ?R s t) (λ (ss, ts).
list-all2 (is-mstep R) ss ts) ((root-steps-substs ?R s t)!i)
      using list-ex-length by blast
    then obtain ss' ts' where ss'ts':(ss', ts') ∈ set (root-steps-substs ?R s t) list-all2
(is-mstep R) ss' ts'
      using nth-mem by fastforce
    with root-steps-substs-exists obtain l r vl σ τ where lr:(l, r) ∈ set R
      and vl:vl = filter (λx. x ∈ vars-term r) (vars-distinct l)

```

```

and  $l:l \cdot \sigma = s$  and  $r:r \cdot \tau = t$ 
and  $\sigma\tau:(ss', ts') = (\text{map } \sigma \text{ } vl, \text{map } \tau \text{ } vl)$ 
by (smt (verit, best) mem-Collect-eq set-filter)
let  $? \tau = \lambda x. (\text{if } x \in \text{vars-term } r \text{ then } \tau \text{ } x \text{ else } \sigma \text{ } x)$ 
from  $r$  have  $r':r \cdot ? \tau = t$ 
by (smt (verit, del-insts) term-subst-eq)
{ fix  $x$  assume  $x:x \in \text{vars-term } l$ 
  then have  $(\sigma \text{ } x, ? \tau \text{ } x) \in \text{mstep } (\text{set } R)$  proof(cases  $x \in \text{set } vl$ )
    case True
      then obtain  $i$  where  $i:i < \text{length } vl \text{ } vl ! i = x$ 
        using in-set-idx by force
      then have  $i1:i < \text{length } ss'$ 
        using  $\sigma\tau$  by simp
      from  $i$  have  $i2:i < \text{length } ts'$ 
        using  $\sigma\tau$  by simp
      from  $ss'ts'(2)$   $i1$   $i2$  have  $\text{is-mstep } R \text{ } (ss' ! i) \text{ } (ts' ! i)$ 
        using list-all2-nthD by blast
      with assms(2)[OF  $ss'ts'(1)$ ]  $i1$   $i2$  have  $(ss' ! i, ts' ! i) \in \text{mstep } (\text{set } R)$ 
        by auto
      then show  $?thesis$ 
        using  $i$   $\sigma\tau$  by auto
    next
      case False
        with  $vl \text{ } x$  show  $?thesis$  by simp
  qed
}
then show  $?thesis$ 
using mstep.rule[OF  $lr$ ]  $l \text{ } r'$  by force
qed

```

```

lemma is-mstep-mstep:
  assumes  $\text{is-mstep } R \text{ } s \text{ } t$ 
  shows  $(s, t) \in \text{mstep } (\text{set } R)$ 
using assms proof (induct rule: is-mstep.induct)
  case ( $1 \text{ } R \text{ } f \text{ } ss \text{ } g \text{ } ts$ )
    from  $1$  consider  $\text{Fun } f \text{ } ss = \text{Fun } g \text{ } ts$ 
      | (rrstep)  $(\text{Fun } f \text{ } ss, \text{Fun } g \text{ } ts) \in \text{rrstep } (\text{set } R)$ 
      | (root)  $\text{list-ex } (\lambda (ss, ts). \text{list-all2 } (\text{is-mstep } R) \text{ } ss \text{ } ts) \text{ } (\text{root-steps-substs } (\text{remove-trivial-rules } R) \text{ } (\text{Fun } f \text{ } ss) \text{ } (\text{Fun } g \text{ } ts))$ 
      | (args)  $f = g$  and  $\text{length } ss = \text{length } ts$  and  $\text{list-all2 } (\text{is-mstep } R) \text{ } ss \text{ } ts$ 
      by (auto split: if-splits)
    then show  $?case$  proof(cases)
      case root
        show  $?thesis$  using mstep-root-helper[OF root]  $1(1)$  by simp
    next
      case args
        { fix  $i$ 
          assume  $i:i < \text{length } ss$ 

```

```

    then have  $si:ss ! i \in \text{set } ss$  by auto
    from  $i \text{ args}(2)$  have  $ti:ts ! i \in \text{set } ts$  by auto
    from  $\text{args}(3)$  have  $is\text{-}mstep\ R\ (ss ! i)\ (ts ! i)$  using  $i \text{ list-all2-nthD}$  by blast
    with  $1(2)[\text{of } ss ! i\ ts ! i]\ \text{args}(1,2)$  si ti have  $(ss ! i, ts ! i) \in mstep\ (\text{set } R)$ 
    by auto
  }
  then show ?thesis using  $\text{args}(1,2)$ 
  by (simp add: mstep.args)
qed (simp-all add: rstep-imp-rstep rstep-imp-mstep)
next
case (2-1  $R\ v\ t$ )
from 2-1 consider  $\text{Var } v = t$ 
|  $(\text{Var } v, t) \in rstep\ (\text{set } R)$ 
|  $(\text{root}) \text{ list-ex } (\lambda (ss, ts). \text{list-all2 } (is\text{-}mstep\ R)\ ss\ ts)\ (\text{root-steps-substs } (\text{remove-trivial-rules } R)\ (\text{Var } v)\ t)$ 
by auto
then show ?case proof(cases)
case root
show ?thesis using mstep-root-helper[OF root] 2-1(1) by simp
qed (simp-all add: rstep-imp-rstep rstep-imp-mstep)
next
case (2-2  $R\ s\ v$ )
from 2-2 consider  $s = \text{Var } v$ 
|  $(s, \text{Var } v) \in rstep\ (\text{set } R)$ 
|  $(\text{root}) \text{ list-ex } (\lambda (ss, ts). \text{list-all2 } (is\text{-}mstep\ R)\ ss\ ts)\ (\text{root-steps-substs } (\text{remove-trivial-rules } R)\ s\ (\text{Var } v))$ 
by auto
then show ?case proof(cases)
case root
show ?thesis using mstep-root-helper[OF root] 2-2(1) by simp
qed (simp-all add:rstep-imp-rstep rstep-imp-mstep)
qed

```

lemma $is\text{-}mstep[simp]$:

$is\text{-}mstep\ R\ s\ t \longleftrightarrow (s, t) \in mstep\ (\text{set } R)$

using $is\text{-}mstep\text{-}mstep\ mstep\text{-}is\text{-}mstep$ by blast

lemma $mstep\text{-}code[code\text{-}unfold]$: $(s, t) \in mstep\ (\text{set } R) \longleftrightarrow is\text{-}mstep\ R\ s\ t$ by simp

fun $\text{root-subst-with-rhs} :: ('f, 'v)\ \text{rules} \Rightarrow ('f, 'v)\ \text{term} \Rightarrow (('f, 'v)\ \text{term} \times ('f, 'v)\ \text{term list})\ \text{list}$

where

$\text{root-subst-with-rhs } R\ s = \text{concat } (\text{map } (\lambda (l, r). \\ \text{(case match } s\ l\ \text{of} \\ \text{None} \Rightarrow [] \\ | \text{Some } \sigma \Rightarrow [(r, \text{map } \sigma\ (\text{vars-distinct } r))])))) \\ R)$

lemma $\text{root-steps-subst-rhs-exists}$:

```

assumes  $(r, ss) \in \text{set } (\text{root-subst-with-rhs } R \ s)$ 
shows  $\exists l \ \sigma. (l, r) \in \text{set } R \wedge l \cdot \sigma = s \wedge ss = \text{map } \sigma \ (\text{vars-distinct } r)$ 
proof –
  from assms obtain  $l$  where  $lr:(l, r) \in \text{set } R \ (r, ss) \in \text{set } (\text{case match } s \ l \ \text{of}$ 
     $\text{None} \Rightarrow []$ 
     $| \text{Some } \sigma \Rightarrow [(r, \text{map } \sigma \ (\text{vars-distinct } r))])$ 
  by auto
  then obtain  $\sigma$  where  $\sigma:\text{match } s \ l = \text{Some } \sigma$ 
  by fastforce
  with  $lr$  show ?thesis
  using match-matches by force
qed

context
  fixes  $R :: ('f, 'v) \text{ rules}$ 
  assumes  $wf\text{-trs } (\text{set } R)$ 
begin

private lemma *:  $\text{list-all } (\lambda(l, r). \text{is-Fun } l \wedge (\text{vars-term } r \subseteq \text{vars-term } l)) \ R$ 
  using  $\langle wf\text{-trs } (\text{set } R) \rangle$  unfolding wf-trs-def by  $(\text{auto simp: list-all-iff})$ 

lemma varcond:
   $\bigwedge l \ r. (l, r) \in \text{set } R \implies \text{is-Fun } l \wedge \text{vars-term } r \subseteq \text{vars-term } l$ 
  using * Ball-set-list-all case-prodD by  $(\text{metis } (\text{no-types, lifting}))$ 

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  $\text{size } (\sigma \ x) < \text{Suc } (\text{size-list size } ts)$ 
  using assms size-match-subst-Fun varcond
  by  $(\text{metis } (\text{no-types, lifting}) \text{ add.right-neutral add-Suc-right subsetD term.size(4)})$ 

fun mstep-rewrite-main ::  $('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term list}$ 
where
   $\text{mstep-rewrite-main } (\text{Var } x) = [\text{Var } x]$ 
   $| \text{mstep-rewrite-main } (\text{Fun } f \ ss) = \text{remdups } ($ 
     $\text{concat } (\text{map } (\lambda(r, ts).$ 
       $(\text{map } (\lambda \text{args}. r \cdot (\text{mk-subst Var } (\text{zip } (\text{vars-distinct } r) \ \text{args}))) \ (\text{product-lists}$ 
         $(\text{map mstep-rewrite-main } ts))))$ 
       $(\text{root-subst-with-rhs } R \ (\text{Fun } f \ ss))))$ 
     $\ @(\text{map } (\lambda ss. \text{Fun } f \ ss) \ (\text{product-lists } (\text{map mstep-rewrite-main } ss))))$ 

lemma mstep-rewrite-main-mstep:
  assumes  $t \in \text{set } (\text{mstep-rewrite-main } s)$ 
  shows  $(s, t) \in \text{mstep } (\text{set } R)$ 
using assms
proof  $(\text{induct } s \ \text{arbitrary: } t \ \text{rule:subterm-induct})$ 

```

```

case (subterm s)
then show ?case proof(cases s)
  case (Var x)
  with subterm(2) show ?thesis by simp
next
  case (Fun f ss)
  with subterm consider (root) t ∈ set (concat (map ( $\lambda(r,ts).$ (map ( $\lambda$ args. r ·
(mk-subst Var (zip (vars-distinct r) args)))
  (product-lists (map mstep-rewrite-main ts)))) (root-subst-with-rhs R (Fun
f ss))))
  | (args) t ∈ set (map ( $\lambda$ ss. Fun f ss) (product-lists (map mstep-rewrite-main
ss)))
  by force
  then show ?thesis
  proof (cases)
  case root
  then obtain r ts where rhs-subst:(r,ts) ∈ set (root-subst-with-rhs R (Fun f
ss))
  t ∈ set (map ( $\lambda$ args. r · (mk-subst Var (zip (vars-distinct
r) args))) (product-lists (map mstep-rewrite-main ts)))
  by force
  from root-steps-subst-rhs-exists[OF rhs-subst(1)] obtain l σ where lr:(l, r)
  ∈ set R
  and sigma:l · σ = Fun f ss ts = map σ (vars-distinct r) by auto
  from rhs-subst(2) obtain args where args:t = r · (mk-subst Var (zip
(vars-distinct r) args))
  args ∈ set (product-lists (map mstep-rewrite-main ts))
  by auto
  then have len:length args = length ts
  using in-set-product-lists-length by fastforce
  then have len':length args = length (vars-distinct r)
  by (simp add: sigma(2))
  let ?τ= $\lambda x.$  if x ∈ vars-term r then (mk-subst Var (zip (vars-distinct r) args))
x else σ x
  from args(1) have t:t = r · ?τ
  by (simp add: term-subst-eq-conv)
  { fix x
  assume x:x ∈ vars-term l
  have (σ x, ?τ x) ∈ mstep (set R) proof(cases x ∈ vars-term r)
  case True
  then obtain i where i:i < length (vars-distinct r) x = vars-distinct r ! i
  by (metis in-set-idx set-vars-term-list vars-term-list-vars-distinct)
  with True len' have tau-x: ?τ x = args!i
  by (simp add: mk-subst-distinct)
  from i sigma(2) have sigma-x:σ x = ts!i
  by simp
  have σ x <math>\triangleleft</math> Fun f ss
  by (metis is-VarI lr sigma(1) subst-image-subterm term.set-cases(2)
varcond x)

```

```

      with sigma-x have ts!i < Fun f ss by simp
      moreover have args!i ∈ set (mstep-rewrite-main (ts!i)) using args(2)
i(1) len' len
      unfolding product-lists-set list-all2-conv-all-nth by force
      ultimately show ?thesis using subterm(1) sigma-x tau-x unfolding Fun
by presburger
      next
      case False
      then show ?thesis by simp
      qed
    }
    then show ?thesis using mstep.intros(3)[OF lr] sigma(1) unfolding Fun t
      by fastforce
  next
  case args
  then obtain ts where t:t = Fun f ts and ts:ts ∈ set (product-lists (map
mstep-rewrite-main ss))
    by auto
  then have len:length ss = length ts using in-set-product-lists-length by force
  { fix i
    assume i:i < length ts
    have ts ! i ∈ set (mstep-rewrite-main (ss ! i))
      using ts[unfolded product-lists-set[of - ss]]
    by (auto simp: list-all2-map2[of (λx ys. x ∈ set ys)] intro: list-all2-nthD[OF
- i])
    with subterm len i have (ss ! i, ts ! i) ∈ mstep (set R)
      unfolding Fun by auto
    }
  with mstep.intros(2) len t Fun show ?thesis
    by metis
  qed
qed
qed
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) wfTRS = {R :: ('f, 'v) rules. wf-trs (set R)}
  by (intro exI[of - Nil], auto simp: wf-trs-def)

```

```

setup-lifting type-definition-wfTRS

```

```

lift-definition get-TRS :: ('f, 'v) wfTRS ⇒ ('f, 'v) rules is λ R. R .

```

```

lemma is-wf-get-TRS: wf-trs (set (get-TRS R'))
  by (transfer, auto)

```

```

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
 $\lambda R.$ if *isOk* (*check-wf-trs* *R*) then *Some* *R* else *None*
by (*force simp: wf-trs-def list.pred-set split: prod.splits*)

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"*)
($\lambda _.$ *mstep-dummy-impl* *R*)
by *simp*

lift-definition (*code-dt*) *get-wfTRS-sub* :: ('f :: showl, 'v :: showl) rules \Rightarrow ('f, 'v)
wfTRS is
 $\lambda R.$ if *isOk* (*check-wf-trs* *R*) then *R* else *Code.abort* *err-wf* ($\lambda _.$ [])
by (*auto simp: wf-trs-def*)

definition *mstep-rewrite* *R* = *mstep-rewrite-wf* (*get-wfTRS-sub* *R*)

lemma *mstep-rewrite-mstep*:
assumes *t* \in *set* (*mstep-rewrite* *R s*)
shows (*s, t*) \in *mstep* (*set* *R*)
proof –
define *R'* **where** *R'* = *get-wfTRS-sub* *R*
have *wf: wf-trs* (*set* (*get-TRS* *R'*))
by (*transfer, auto*)
have *sub: set* (*get-TRS* *R'*) \subseteq *set* *R* **unfolding** *R'-def* **by** (*transfer, auto*)
from *mstep-rewrite-main-mstep*[*OF wf, folded mstep-rewrite-wf-def, OF assms(1)*][*unfolded*
mstep-rewrite-def, folded R'-def]
have (*s, t*) \in *mstep* (*set* (*get-TRS* *R'*)) .
with *mstep-mono*[*OF sub*] **show** *?thesis* **by** *auto*
qed

end