Proof Terms for Term Rewriting

Christina Kirk (Kohl)

University of Innsbruck, Austria

February 6, 2026

Abstract

Proof terms are first-order terms that represent reductions in term rewriting. They were initially introduced in [6] and [5, Chapter 8] by van Oostrom and de Vrijer to study equivalences of reductions in left-linear rewrite systems. This entry formalizes proof terms for multi-steps in first-order term rewrite systems. We define simple proof terms (i.e., without a composition operator) and establish the correspondence to multi-steps: each proof term represents a multi-step with the same source and target, and every multi-step can be expressed as a proof term. The formalization moreover includes operations on proof terms, such as residuals, join, and deletion and a method for labeling proof term sources to identify overlaps between two proof terms.

This formalization is part of the *Isabelle Formalization of Rewriting* IsaFoR and is an essential component of several formalized confluence and commutation results involving multi-steps [2, 3, 4, 1].

Contents

1 Preliminaries **2**
- 1.1 Utilities for Lists . . . 2
  - 1.1.1 Lists of *option* . . . 8
- 1.2 Results About Linear Terms . . . 9
- 1.3 Results About Substitutions and Contexts . . . 12
  - 1.3.1 Utilities for *mk-subst* . . . 17
- 1.4 Matching Terms . . . 19
  - 1.4.1 Matching of Linear Terms . . . 21

2 Proof Terms **25**
- 2.1 Definitions . . . 25
- 2.2 Frequently Used Locales/Contexts . . . 27
- 2.3 Proof Term Predicates . . . 28
- 2.4 'Normal' Terms vs. Proof Terms . . . 32

2.5 Substitutions . . . 33
2.6 Contexts . . . 37
2.7 Source and Target . . . 39
2.8 Additional Results . . . 49
2.9 Proof Terms Represent Multi-Steps . . . 51

3 Operations on Proof Terms **53**
3.1 Residuals . . . 57
3.2 Join . . . 73
3.2.1 N-Fold Join . . . 89
3.3 Deletion . . . 125
3.4 Computations With Single Redexes . . . 128

4 Orthogonal Proof Terms **138**

5 Labels and Overlaps **150**
5.1 Labeled Proof Terms . . . 150
5.2 Measuring Overlap . . . 199
5.3 Collecting Overlapping Positions . . . 208

6 Redex Patterns **212**

1 Preliminaries

theory *Proof-Term-Utils*
imports
First-Order-Terms.Matching
First-Order-Terms.Term-Impl
begin

1.1 Utilities for Lists

lemma *obtain-list-with-property*:
assumes $\forall\, x \in \mathit{set}\ \mathit{xs}.\ \exists\, a.\ P\ a\ x$
shows $\exists\, \mathit{as}.\ \mathit{length}\ \mathit{as} = \mathit{length}\ \mathit{xs} \wedge (\forall\, i < \mathit{length}\ \mathit{xs}.\ P\ (\mathit{as}!i)\ (\mathit{xs}!i))$
using *assms* **proof**(*induct xs*)
case (*Cons a xs*)
then show *?case*
by (*metis length-map nth-map nth-mem*)
qed *simp*

lemma *card-Union-Sum*:
assumes *is-partition* (*map f* [*0*..<*length xs*])
and $\forall\, i < \mathit{length}\ \mathit{xs}.\ \mathit{finite}\ (f\ i)$
shows $\mathit{card}\ (\bigcup i{<}\mathit{length}\ \mathit{xs}.\ f\ i) = (\sum i{<}\mathit{length}\ \mathit{xs}.\ \mathit{card}\ (f\ i))$
proof−
from *assms*(*1*) **have** *disj*:*pairwise* ($\lambda s\ t.\ \mathit{disjnt}\ (f\ s)\ (f\ t)$) {..<*length xs*}

unfolding *pairwise-def is-partition-alt is-partition-alt-def disjnt-def* **by** *simp*
then have *pairwise disjnt* (*f* ‘ {..<*length xs*})
by (*metis* (*mono-tags*, *lifting*) *pairwiseD pairwise-imageI*)
then have *card* (⋃ *i*<*length xs*. *f i*) = *sum card* (*f* ‘ {..<*length xs*})
using *assms*(*2*) *card-Union-disjoint* **by** (*metis* (*mono-tags*, *lifting*) *imageE lessThan-iff*)
with *disj* **show** *?thesis*
using *sum-card-image* **by** (*metis finite-lessThan*)
qed

lemma *sum-sum-concat*: (∑ *i*<*length xs*. ∑ *x*←*f* (*xs*!*i*). *g x*) = (∑ *x*←*concat* (*map f xs*). *g x*)
proof(*induct xs*)
case (*Cons a xs*)
then show *?case* **unfolding** *list.map concat.simps map-append sum-list-append*
by (*metis* (*mono-tags*, *lifting*) *length-nth-simps*(*2*) *nth-Cons-0 nth-Cons-Suc sum.cong sum.lessThan-Suc-shift*)
qed *simp*

lemma *concat-map2-zip*:
assumes *length xs* = *length ys*
and ∀ *i* < *length xs*. *length* (*xs*!*i*) = *length* (*ys*!*i*)
shows *concat* (*map2 zip xs ys*) = *zip* (*concat xs*) (*concat ys*)
using *assms* **proof**(*induct xs arbitrary*:*ys rule*:*rev-induct*)
case (*snoc x xs*)
from *snoc*(*2*) **obtain** *y ys*′ **where** *y*:*ys* = *ys*′@[*y*]
by (*metis append-is-Nil-conv length-0-conv neq-Nil-conv rev-exhaust*)
moreover with *snoc*(*2*) **have** *l*:*length xs* = *length ys*′ **by** *simp*
moreover with *snoc*(*3*) **have** *l*′:∀ *i* < *length xs*. *length* (*xs*!*i*) = *length* (*ys*′!*i*)
unfolding *y* **by** (*metis* (*no-types*, *lifting*) *Ex-less-Suc add-Suc-right append.right-neutral append-Cons-nth-left length-Cons length-append order-less-trans*)
ultimately have *IH*:*concat* (*map2 zip xs ys*′) = *zip* (*concat xs*) (*concat ys*′)
using *snoc*(*1*) **by** *presburger*
have ∗:*concat* (*map2 zip* (*xs* @ [*x*]) *ys*) = *concat* (*map2 zip xs ys*′) @ (*zip x y*)
unfolding *y zip-append*[*OF l*] **by** *simp*
have *length* (*concat xs*) = *length* (*concat ys*′)
using *l l*′ *eq-length-concat-nth* **by** *blast*
then show *?case*
unfolding ∗ *IH* **unfolding** *y concat-append* **using** *zip-append* **by** *simp*
qed *simp*

lemma *sum-list-less*:
assumes *less*:*i* < *j*
and *i*′*j*′:*i*′ < *length xs j*′ < *length xs*
and *j*′′:*j*′′ < *length* (*xs*!*j*′)
and *sums*:*i* = *sum-list* (*map length* (*take i*′ *xs*)) + *i*′′ *j* = *sum-list* (*map length* (*take j*′ *xs*)) + *j*′′
shows *i*′ ≤ *j*′
proof(*rule ccontr*)

assume $*$:$\neg$ $i' \leq j'$
then have *subsums*:*sum-list* (*map length* (*take i' xs*)) = *sum-list* (*map length* (*take j' xs*)) + *sum-list* (*map length* (*take* (*i'*−*j'*) (*drop j' xs*)))
by (*metis le-add-diff-inverse map-append nat-le-linear sum-list-append take-add*)

from $*$ **have** *take* (*i'* − *j'*) (*drop j' xs*) = *xs*!*j'* # (*take* (*i'* − (*Suc j'*)) (*drop* (*Suc j'*) *xs*))
using *i'j'* **by** (*metis Cons-nth-drop-Suc Suc-diff-Suc linorder-le-less-linear take-Suc-Cons*)

with *j''* **have** *j''* < *sum-list* (*map length* (*take* (*i'*−*j'*) (*drop j' xs*))) **by** *simp*
then show *False*
using *sums subsums less* **by** *linarith*
qed

lemma *zip-symm*: (*x*, *y*) ∈ *set* (*zip xs ys*) ⟹ (*y*, *x*) ∈ *set* (*zip ys xs*)
by (*induct xs ys rule*:*list-induct2'*) *auto*

lemma *sum-list-elem*:
(∑ *x*←[*y*]. *f x*) = *f y*
by *simp*

lemma *sum-list-zero*:
assumes ∀ *i* < *length xs*. *f* (*xs*!*i*) = *0*
shows (∑ *x*←*xs*. *f x*) = *0*
by (*metis assms map-eq-conv' monoid-add-class.sum-list-0*)

lemma *distinct-is-partition*:
assumes *distinct* (*concat ts*)
shows *is-partition* (*map set ts*)
using *assms* **proof**(*induct ts*)
case *Nil*
then show *?case*
using *is-partition-Nil* **by** *auto*
next
case (*Cons t ts*)
{fix *i j* **assume** *j*:*j* < *length* (*t*#*ts*) **and** *ij*:*i* < *j*
have (*map set* (*t*#*ts*))!*i* ∩ (*map set* (*t*#*ts*))!*j* = {} **proof**(*cases i*)
case *0*
show *?thesis* **using** *Cons*(*2*) **unfolding** *0*
using *ij j* **by** *force*
next
case (*Suc n*)
from *Cons* **have** *is-partition* (*map set ts*) **by** *simp*
then show *?thesis*
unfolding *Suc is-partition-def* **using** *j ij* **using** *Suc Suc-less-eq2* **by** *fastforce*
qed
}
then show *?case* **unfolding** *is-partition-def* **by** *simp*
qed

lemma *filter-ex-index*:
assumes $x = filter\ f\ xs\ !\ i\ i < length\ (filter\ f\ xs)$
shows $\exists j.\ j < length\ xs \wedge x = xs!j$
using *assms* **proof**(*induct xs arbitrary*:*i*)
case (*Cons y ys*)
show *?case* **proof**(*cases f y*)
case *True*
then have *filter*:*filter f* (*y*#*ys*) = *y*#(*filter f ys*) **by** *simp*
show *?thesis* **proof**(*cases i*)
case *0*
from *Cons*(*2*) **show** *?thesis* **unfolding** *filter 0* **by** *auto*
next
case (*Suc n*)
from *Cons*(*2*) **have** $x = filter\ f\ ys\ !\ n$
unfolding *Suc filter* **by** *simp*
moreover from *Cons*(*3*) **have** $n < length\ (filter\ f\ ys)$
unfolding *Suc filter* **by** *simp*
ultimately obtain *j* **where** *j*<*length ys* **and** $x = ys\ !\ j$
using *Cons*(*1*) **by** *blast*
then show *?thesis* **by** *auto*
qed
next
case *False*
then have *filter*:*filter f* (*y*#*ys*) = *filter f ys* **by** *simp*
from *Cons* **obtain** *j* **where** *j*<*length ys* **and** $x = ys\ !\ j$
unfolding *filter* **by** *blast*
then show *?thesis* **by** *auto*
qed
qed *simp*

lemma *filter-index-neq'*:
assumes $i < j\ j < length\ (filter\ f\ xs)$
shows $\exists\ i'\ j'.\ i' < length\ xs \wedge j' < length\ xs \wedge i' < j' \wedge xs\ !\ i' = (filter\ f\ xs)\ !\ i \wedge xs\ !\ j' = (filter\ f\ xs)\ !\ j$
using *assms* **proof**(*induct xs arbitrary*: *i j*)
case (*Cons x xs*)
then show *?case* **proof**(*cases f x*)
case *True*
show *?thesis* **proof**(*cases i*)
case *0*
then have *i0*:*filter f* (*x*#*xs*) ! *i* = (*x*#*xs*) ! *0*
using ‹*f x*› **by** *simp*
from *Cons*(*2*) **obtain** j' **where** $j = Suc\ j'$
unfolding *0* **using** *gr0-implies-Suc* **by** *blast*
with *Cons*(*3*) **have** $j' < length\ (filter\ f\ xs)$
unfolding *filter.simps* **using** ‹*f x*› **by** *simp*
then obtain j'' **where** j'':$j'' < length\ xs\ filter\ f\ xs\ !\ j' = xs\ !\ j''$
by (*meson filter-ex-index*)

```
    then have filter f (x#xs) ! j = (x#xs) ! (Suc j'')
      using ‹f x› ‹j = Suc j'› by simp
    with i0 j''(1) show ?thesis
      by (metis length-nth-simps(2) not-less-eq zero-less-Suc)
  next
    case (Suc i')
    from Cons(2) obtain j' where j:j = Suc j'
      unfolding Suc using Suc-lessE by auto
    from Cons(1)[of i' j'] Cons(2,3) obtain i'' j'' where i'' < length xs j'' <
length xs i'' < j'' xs ! i'' = filter f xs ! i' xs ! j'' = filter f xs ! j'
      using Suc True j by auto
    then show ?thesis
      by (smt (verit) Suc Suc-less-eq True filter.simps(2) j length-nth-simps(2)
nth-Cons-Suc)
  qed
 next
  case False
  then have filter f (x#xs) = filter f xs by simp
  with Cons show ?thesis
    by (metis Suc-less-eq length-nth-simps(2) nth-Cons-Suc)
 qed
qed simp

lemma filter-index-neq:
 assumes i ≠ j i < length (filter f xs) j < length (filter f xs)
 shows ∃ i' j'. i' < length xs ∧ j' < length xs ∧ i' ≠ j' ∧ xs ! i' = (filter f xs) !
i ∧ xs ! j' = (filter f xs) ! j
using assms filter-index-neq' proof(cases i < j)
 case False
 then have ∗:j < i using assms(1) by simp
 then show ?thesis using filter-index-neq'[OF ∗ assms(2)] by blast
qed blast

lemma nth-drop-equal:
 assumes length xs = length ys
   and ∀ j < length xs. j ≥ i ⟶ xs!j = ys!j
 shows drop i xs = drop i ys
using assms proof (induct i arbitrary: xs ys)
 case 0
 then show ?case
   using nth-equalityI by blast
next
 case (Suc i)
 then show ?case proof(cases Suc i < length xs)
   case True
   then obtain x xs' where x:xs = x # xs'
     by (metis Suc-length-conv Suc-lessE)
   with Suc(2) obtain y ys' where y:ys = y # ys'
     by (metis length-greater-0-conv nth-drop-0)
```

```
    from Suc(1)[of xs' ys'] have drop i xs' = drop i ys'
      using Suc(2,3) unfolding x y
    by (metis Suc-le-mono length-nth-simps(2) linorder-not-le nat.inject nth-Cons-Suc)
    then show ?thesis unfolding x y by simp
  qed simp
qed

lemma union-take-drop-list:
  assumes i < length xs
  shows (set (take i xs)) ∪ (set (drop (Suc i) xs)) = {xs!j | j. j < length xs ∧ j ≠
i}
proof−
  from assms have i:i ≤ length xs by simp
  have set1:set (take i xs) = {xs ! j |j. j < i}
    using nth-image[OF i] unfolding image-def by fastforce
  from assms have i:Suc i ≤ length xs by simp
  have set2:set (drop (Suc i) xs) = {xs !j |j. i < j ∧ j < length xs} proof
    {fix x assume x ∈ set (drop (Suc i) xs)
      then have x ∈ {xs !j |j. i < j ∧ j < length xs}
        unfolding set-conv-nth nth-drop[OF i] length-drop by auto
    }
    then show set (drop (Suc i) xs) ⊆ {xs ! j |j. i < j ∧ j < length xs} by auto
    {fix x assume x ∈ {xs !j |j. i < j ∧ j < length xs}
      then have x ∈ set (drop (Suc i) xs)
        unfolding set-conv-nth nth-drop[OF i] length-drop
      by (smt (verit, best) Suc-leI add-diff-inverse-nat i mem-Collect-eq nat-add-left-cancel-less
not-less-eq-eq)
    }
    then show {xs ! j |j. i < j ∧ j < length xs} ⊆ set (drop (Suc i) xs) by auto
  qed
  {fix x assume x ∈ set (take i xs) ∪ set (drop (Suc i) xs)
    then consider x ∈ set (take i xs) | x ∈ set (drop (Suc i) xs) by blast
    then have x ∈ {xs ! j |j. j < length xs ∧ j ≠ i} proof(cases)
      case 1
      with set1 show ?thesis using in-set-idx by fastforce
    next
      case 2
      with set2 show ?thesis using in-set-idx by fastforce
    qed
  }moreover
  {fix x assume x ∈ {xs ! j |j. j < length xs ∧ j ≠ i}
    then obtain j where x = xs!j and j:j < length xs j ≠ i
      by blast
    then have x ∈ set (take i xs) ∪ set (drop (Suc i) xs)
      using set1 set2 using nat-neq-iff by auto
  }
  ultimately show ?thesis by auto
qed
```

lemma *list-tl-eq*:
assumes *length xs = length ys xs ≠ []*
and *∀ i < length xs. i > 0 ⟶ xs!i = ys!i*
shows *tl xs = tl ys*
by (*metis Suc-le-lessD assms(1) assms(3) length-greater-0-conv list.sel(3) nth-drop-0 nth-drop-equal*)

1.1.1 Lists of *option*

lemma *length-those*:
assumes *those xs = Some ys*
shows *length xs = length ys*
using *assms* **proof**(*induction xs arbitrary:ys*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a xs*)
from *Cons(2)* **obtain** *ys′* **where** *ys′:those xs = Some ys′*
by (*smt not-None-eq option.case-eq-if option.simps(8) those.simps(2)*)
from *Cons(2)* **obtain** *y* **where** *y:Some y = a*
by (*metis option.case-eq-if option.exhaust-sel option.simps(3) those.simps(2)*)
from *y ys′* **have** *those* (*Cons a xs*) = *Some* (*Cons y ys′*)
by *auto*
then show *?case* **using** *Cons ys′*
by *auto*
qed

lemma *those-not-none-x*: *those xs = Some ys ⟹ x ∈ set xs ⟹ x ≠ None*
proof (*induction xs arbitrary*: *x ys*)
case (*Cons a xs*)
from *Cons(2)* **have** *a ≠ None* **using** *option.simps(4)* **by** *fastforce*
from *this Cons(2)* **have** *those xs ≠ None* **by** *auto*
then show *?case* **using** *Cons(1,3)* ‹*a ≠ None*› **by** *auto*
qed (*simp*)

lemma *those-not-none-xs*: *list-all* (*λx. x ≠ None*) *xs ⟹ those xs ≠ None*
by (*induction xs*) *auto*

lemma *those-some*:
assumes *length xs = length ys ∀ i < length xs. xs!i = Some* (*ys!i*)
shows *those xs = Some ys*
using *assms* **by** (*induct rule:list-induct2*) (*simp*, *force*)

lemma *those-some2*:
assumes *those xs = Some ys*
shows *∀ i < length xs. xs!i = Some* (*ys!i*)
proof−
from *assms* **have** *length xs = length ys* **by** (*simp add*: *length-those*)
then show *?thesis* **using** *assms* **proof**(*induction xs ys rule:list-induct2*)

case (*Cons x xs y ys*)
from *Cons*(*3*) **have** $x \neq None$ **by** (*metis list.set-intros*(*1*) *those-not-none-x*)
with *Cons*(*3*) **have** $*{:}x = Some\ y$ **by** *force*
with *Cons*(*3*) **have** *those xs* = *Some ys* **by** *force*
with $*$ *Cons*(*2*) **show** *?case* **by** (*simp add*: *nth-Cons'*)
qed *simp*
qed

lemma *exists-some-list*:
assumes $\forall\, i < length\ xs.\ (\exists\, y.\ xs!i = Some\ y)$
shows $\exists\ ys.\ (\forall\, i < length\ xs.\ xs!i = Some\ (ys!i)) \wedge length\ ys = length\ xs$
by (*metis assms length-map nth-map option.sel*)

1.2 Results About Linear Terms

lemma *linear-term-var-vars-term-list*:
assumes *linear-term t*
shows *vars-term-list t* = *vars-distinct t*
using *assms linear-term-distinct-vars*
by (*metis comp-apply distinct-rev remdups-id-iff-distinct rev-rev-ident*)

lemma *linear-term-unique-vars*:
assumes *linear-term s*
and $p \in poss\ s$ **and** $s|{-}p = Var\ x$
and $q \in poss\ s$ **and** $s|{-}q = Var\ x$
shows $p = q$
proof(*rule ccontr*)
assume $p \neq q$
with *assms*(*2*−) **obtain** *i j* **where** *ij*:$i < length\ (var\text{-}poss\text{-}list\ s)\ j < length\ (var\text{-}poss\text{-}list\ s)\ i \neq j$
var-poss-list s ! *i* = *p var-poss-list s* ! *j* = *q*
by (*metis in-set-idx var-poss-iff var-poss-list-sound*)
with *assms*(*3*,*5*) **have** *vars-term-list s* ! *i* = *vars-term-list s* ! *j*
by (*metis length-var-poss-list term.inject*(*1*) *vars-term-list-var-poss-list*)
moreover from *assms*(*1*) **have** *distinct* (*vars-term-list s*)
by (*metis distinct-remdups distinct-rev linear-term-var-vars-term-list o-apply*)
ultimately show *False* **using** *ij*(*1*,*2*,*3*)
by (*metis distinct-Ex1 length-var-poss-list nth-mem*)
qed

lemma *linear-term-ctxt*:
assumes *linear-term t*
and $p \in poss\ t$
shows *vars-ctxt* (*ctxt-of-pos-term p t*) ∩ *vars-term* (*t*|*-p*) = {}
using *assms* **proof**(*induct p arbitrary*:*t*)
case (*Cons i p*)
from *Cons*(*3*) **obtain** *f ts* **where** *t*:$t = Fun\ f\ ts\ i < length\ ts\ p \in poss\ (ts!i)$
using *args-poss* **by** *blast*
with *Cons*(*1*,*2*) **have** *IH*:*vars-ctxt* (*ctxt-of-pos-term p* (*ts*!*i*)) ∩ *vars-term* ((*ts*!*i*)

```
|- p) = {}
    by simp
  {fix j assume j:j < length ts j ≠ i
    with Cons(2) have vars-term (ts!j) ∩ vars-term (ts!i |-p) = {}
    unfolding t using var-in-linear-args t(2,3) by (metis (no-types, opaque-lifting)
Int-Un-distrib disjoint-iff sup-bot.neutr-eq-iff vars-ctxt-pos-term)
  }
  then have ⋃ {vars-term (ts ! j) |j. j < length ts ∧ j ≠ i} ∩ vars-term (ts!i |-p)
= {}
    by blast
  moreover have (⋃ (vars-term ‘ set (take i ts)) ∪ ⋃ (vars-term ‘ set (drop (Suc
i) ts))) =
                ⋃ {vars-term (ts ! j) |j. j < length ts ∧ j ≠ i}
   unfolding set-map[symmetric] take-map[symmetric] drop-map[symmetric] Union-Un-distrib[symmetric]
   using union-take-drop-list[where xs=(map vars-term ts)] unfolding length-map
using t(2) by auto
  ultimately show ?case unfolding t ctxt-of-pos-term.simps subt-at.simps using
IH
    by (metis (no-types, lifting) bot-eq-sup-iff inf-sup-distrib2 vars-ctxt.simps(2))

qed simp

lemma linear-term-obtain-subst:
  assumes linear-term (Fun f ts) and l:length ts = length ss
    and substs: ∀ i< length ts. (∃ σ. ts!i · σ = ss!i)
  shows ∃ σ. Fun f ts · σ = Fun f ss
  using assms proof(induct ts arbitrary: ss)
  case (Cons t ts)
  from Cons(3) obtain s ss' where ss:ss = s#ss'
    by (metis length-Suc-conv)
  from Cons(2) have lin:linear-term (Fun f ts)
    unfolding linear-term.simps by (simp add: is-partition-Cons)
  from Cons(4) have ∀ i<length ts. ∃ σ. ts ! i · σ = ss' ! i
    unfolding ss by (metis length-nth-simps(2) not-less-eq nth-Cons-Suc)
  then obtain σ where σ:Fun f ts · σ = Fun f ss'
    using Cons(1)[OF lin, of ss'] using Cons.prems(2) ss by auto
  from Cons(4) obtain σ1 where σ1:t · σ1 = s
    using ss by auto
  let ?σ=λx. if x ∈ vars-term t then σ1 x else σ x
  have t:t · ?σ = s
    by (simp add: σ1 term-subst-eq)
  {fix i assume i < length ts
    then have ts!i · ?σ = ss'!i
      by (smt (verit, ccfv-SIG) Cons.prems(1) Cons.prems(2) Suc-inject Suc-leI σ
σ1 eval-term.simps(2) le-imp-less-Suc length-nth-simps(2) map-nth-eq-conv nth-Cons-0
nth-Cons-Suc ss term.sel(4) term-subst-eq var-in-linear-args zero-less-Suc)
  }
  with t have Fun f (t#ts) · ?σ = Fun f ss
    using Cons.prems(2) map-nth-eq-conv ss by auto
```

```
  then show ?case by blast
qed simp

lemma linear-ctxt-of-pos-term:
  assumes linear-term t and lin-s:linear-term s and p:p ∈ poss t
    and vars-term t ∩ vars-term s = {}
  shows linear-term (replace-at t p s)
using assms proof(induct t arbitrary:p)
  case (Var x)
  with p have p = [] by simp
  with lin-s show ?case by simp
next
  case (Fun f ts)
  from lin-s show ?case proof(cases p)
    case (Cons i p')
    with Fun(4) have i:i < length ts by simp
    with Fun(4) have p':p' ∈ poss (ts!i) unfolding Cons by simp
    {fix n assume n:n < length ts n ≠ i
      with Fun(2) have vars-term (ts!n) ∩ vars-term (ts!i) = {}
        by (metis disjoint-iff i var-in-linear-args)
      then have vars-term (ts!n) ∩ vars-ctxt (ctxt-of-pos-term p' (ts!i)) = {}
        using p' vars-ctxt-pos-term by fastforce
      moreover from n Fun(5) have vars-term (ts!n) ∩ vars-term s = {}
        by (meson disjoint-iff nth-mem term.set-intros(4))
        ultimately have vars-term (ts!n) ∩ vars-term ((ctxt-of-pos-term p' (ts !
i))⟨s⟩) = {}
        unfolding vars-term-ctxt-apply by blast
    }
    with Fun(2) have is-partition (map vars-term (take i ts @ (ctxt-of-pos-term p'
(ts ! i))⟨s⟩ # drop (Suc i) ts))
      unfolding linear-term.simps is-partition-def by (smt (z3) Int-commute ap-
pend-Cons-nth-not-middle i id-take-nth-drop
      length-append length-map length-nth-simps(2) linorder-neq-iff nth-append-take
nth-map order.strict-implies-order order.strict-trans)
    moreover have linear-term ((ctxt-of-pos-term p' (ts ! i))⟨s⟩)
    using Fun p' by (meson disjoint-iff i linear-term.simps(2) nth-mem term.set-intros(4))

    ultimately show ?thesis
      using Fun(2) unfolding Cons ctxt-of-pos-term.simps intp-actxt.simps lin-
ear-term.simps
     by (metis Un-iff in-set-dropD in-set-takeD set-ConsD set-append)
  qed simp
qed

lemma distinct-vars:
  assumes ⋀p q x y. p ≠ q ⟹ p ∈ poss t ⟹ q ∈ poss t ⟹ t|-p = Var x ⟹
t|-q = Var y ⟹ x ≠ y
  shows distinct (vars-term-list t)
proof−
```

```
  {fix i j assume ij:i ≠ j and i:i < length (vars-term-list t) and j:j < length
(vars-term-list t)
    let ?p=var-poss-list t ! i and ?q=var-poss-list t ! j
    let ?x=vars-term-list t ! i and ?y=vars-term-list t ! j
    from ij i j have pq:?p ≠ ?q
      by (simp add: distinct-var-poss-list length-var-poss-list nth-eq-iff-index-eq)
    have p:?p ∈ poss t
     by (metis i length-var-poss-list nth-mem var-poss-imp-poss var-poss-list-sound)

    have q:?q ∈ poss t
     by (metis j length-var-poss-list nth-mem var-poss-imp-poss var-poss-list-sound)

    have ?x ≠ ?y
      using assms[OF pq p q] i j by (simp add: vars-term-list-var-poss-list)
  }
  then show ?thesis by (meson distinct-conv-nth)
qed

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 t Fun(1) have linear-term t
      by auto
  }
  moreover have is-partition (map vars-term ts)
   using Fun(2) unfolding vars-term-list.simps using distinct-is-partition set-vars-term-list
    by (metis (mono-tags, lifting) length-map map-nth-eq-conv)
  ultimately show ?case by simp
qed simp

lemma distinct-vars-eq-linear: linear-term t = distinct (vars-term-list t)
  using distinct-vars-linear-term linear-term-distinct-vars by blast
```

1.3 Results About Substitutions and Contexts

```
lemma ctxt-apply-term-subst:
  assumes linear-term t and i < length (vars-term-list t)
    and p = (var-poss-list t)!i
  shows (ctxt-of-pos-term p (t · σ))⟨s⟩ = t · σ((vars-term-list t)!i := s)
proof−
  from assms(2,3) have t|-p = Var ((vars-term-list t)!i)
    by (metis vars-term-list-var-poss-list)
  with assms show ?thesis
   by (smt (verit, ccfv-threshold) filter-cong fun-upd-other fun-upd-same length-var-poss-list
```

linear-term-replace-in-subst nth-mem var-poss-imp-poss var-poss-list-sound)
qed

lemma *ctxt-subst-apply*:
assumes $p \in$ *poss* t **and** $t|\text{-}p =$ *Var* x
and *linear-term* t
shows $((\textit{ctxt-of-pos-term}\ p\ t) \cdot_c \sigma)\langle s\rangle = t \cdot \sigma(x := s)$
unfolding *ctxt-of-pos-term-subst*[*symmetric*, *OF assms*(*1*)]
using *assms*
by (*smt* (*verit*) *fun-upd-apply linear-term-replace-in-subst*)

lemma *ctxt-of-pos-term-hole-subst*:
assumes *linear-term* t
and $i <$ *length* (*var-poss-list* t) **and** $p =$ *var-poss-list* $t\ !\ i$
and $\forall x \in$ *vars-term* $t.\ x \neq$ *vars-term-list* $t\ !i \longrightarrow \sigma\ x = \tau\ x$
shows *ctxt-of-pos-term* $p\ (t \cdot \sigma) =$ *ctxt-of-pos-term* $p\ (t \cdot \tau)$
using *assms* **proof**(*induct p arbitrary*: $t\ i$)
case (*Cons j p*)
from *Cons*(*3*,*4*) **have** $j\#p \in$ *var-poss* t
using *nth-mem* **by** *force*
then obtain *f ts* **where** *ts*:$j <$ *length ts* $t =$ *Fun f ts* $p \in$ *var-poss* (*ts*!j)
by (*metis args-poss subt-at.simps*(*2*) *var-poss-iff*)
then obtain i' **where** i':$i' <$ *length* (*var-poss-list* (*ts*!j)) $p =$ *var-poss-list* (*ts*!j)!i'
using *var-poss-list-sound* **by** (*metis in-set-conv-nth*)
from *Cons*(*3*,*4*) **have** *Var* (*vars-term-list* $t\ !\ i) = t|\text{-}(j\#p)$
by (*metis length-var-poss-list vars-term-list-var-poss-list*)
also have ... $= (ts!j)|\text{-}p$
unfolding *ts*(*2*) **by** *simp*
also have ... $=$ *Var* (*vars-term-list* (*ts*!j) $!\ i'$)
using i' **by** (*simp add*: *length-var-poss-list vars-term-list-var-poss-list*)
finally have $*$:*vars-term-list* $t\ !\ i =$ *vars-term-list* (*ts* $!\ j$) $!\ i'$ **by** *simp*
with *Cons*(*5*) **have** $\forall x{\in}$*vars-term* (*ts*!j). $x \neq$ *vars-term-list* (*ts*!j) $!\ i' \longrightarrow \sigma\ x = \tau\ x$
unfolding *ts*(*2*) **using** *ts*(*1*) **by** *auto*
with *Cons*(*2*) i' *ts* **have** *IH*:*ctxt-of-pos-term* p ((*ts*!j) $\cdot\ \sigma$) $=$ *ctxt-of-pos-term* p ((*ts*!j) $\cdot\ \tau$)
using *Cons*(*1*)[*of ts*!j i'] **by** (*meson linear-term.simps*(*2*) *nth-mem*)
{fix j' **assume** j':$j' <$ *length ts* $j' \neq j$
with *Cons*(*2*) **have** *vars-term* (*ts* $!\ j') \cap$ *vars-term* (*ts*!j) $= \{\}$
unfolding *ts*(*2*) **by** (*metis disjoint-iff ts*(*1*) *var-in-linear-args*)
then have $\forall x \in$ *vars-term* (*ts*!j'). $\sigma\ x = \tau\ x$
using *Cons*(*5*) $j' *$ **by** (*metis disjoint-iff i'*(*1*) *length-var-poss-list nth-mem set-vars-term-list term.set-intros*(*4*) *ts*(*2*))
then have (*ts*!j') $\cdot\ \sigma =$ (*ts*!j') $\cdot\ \tau$
by (*meson term-subst-eq*)
} note t'=*this*
with *ts*(*1*) **have** *take j* (*map* ($\lambda t.\ t \cdot \sigma$) *ts*) $=$ *take j* (*map* ($\lambda t.\ t \cdot \tau$) *ts*)
using *nth-take-lemma*[*of j* (*map* ($\lambda t.\ t \cdot \sigma$) *ts*) (*map* ($\lambda t.\ t \cdot \tau$) *ts*)] **by** *simp*
moreover from t' *ts*(*1*) **have** (*drop* (*Suc j*) (*map* ($\lambda t.\ t \cdot \sigma$) *ts*)) $=$ (*drop* (*Suc*

```
j) (map (λt. t · τ) ts))
    using nth-drop-equal[of (map (λt. t · σ) ts) (map (λt. t · τ) ts) Suc j] by auto
  ultimately show ?case
    unfolding ts(2) eval-term.simps ctxt-of-pos-term.simps using IH by (simp
add: ts(1))
qed simp

lemma ctxt-apply-ctxt-apply:
  assumes p ∈ poss t
 shows (ctxt-of-pos-term (p@q) ((ctxt-of-pos-term p t) ⟨s⟩)) ⟨u⟩ = (ctxt-of-pos-term
p t)⟨(ctxt-of-pos-term q s) ⟨u⟩⟩
 by (metis assms ctxt-ctxt ctxt-of-pos-term-append hole-pos-ctxt-of-pos-term hole-pos-id-ctxt
hole-pos-poss replace-at-subt-at)

lemma replace-at-append-subst:
  assumes linear-term t
    and p ∈ poss t t|-p = Var x
  shows (ctxt-of-pos-term (p@q) (t · σ)) ⟨s⟩ = t · σ(x := (ctxt-of-pos-term q (σ
x)) ⟨s⟩)
  using assms proof(induct p arbitrary:t)
  case (Cons i p)
  then obtain f ts where t:t = Fun f ts and i:i < length ts and p:p ∈ poss (ts!i)
    by (meson args-poss)
  from Cons(4) have x:(ts!i)|-p = Var x
    unfolding t by simp
  from Cons(2) have lin:linear-term (ts!i)
    using i t by simp
 have IH:(ctxt-of-pos-term (p@q) ((ts!i) · σ)) ⟨s⟩ = (ts!i) · σ(x := (ctxt-of-pos-term
q (σ x)) ⟨s⟩)
    using Cons(1)[OF lin p x] .
  let ?σ=σ(x := (ctxt-of-pos-term q (σ x)) ⟨s⟩)
  {fix j assume j:j < length ts j ≠ i
    from x have x ∈ vars-term (ts!i)
      by (metis p subsetD term.set-intros(3) vars-term-subt-at)
    then have x ∉ vars-term (ts!j)
      using j Cons(2) unfolding t by (meson i var-in-linear-args)
    then have (ts!j) · σ = (ts!j) · ?σ
      by (simp add: term-subst-eq-conv)
  } note sigma=this
  then have take i (map (λt. t · σ) ts) = take i (map (λt. t · ?σ) ts)
    using nth-take-lemma[of i (map (λt. t · σ) ts) (map (λt. t · ?σ) ts)] i by simp
  moreover from sigma have drop (Suc i) (map (λt. t · σ) ts) = drop (Suc i)
(map (λt. t · ?σ) ts)
    using nth-drop-equal[of (map (λt. t · σ) ts) (map (λt. t · ?σ) ts)] i by simp
  ultimately show ?case
   unfolding t append-Cons eval-term.simps ctxt-of-pos-term.simps intp-actxt.simps
nth-map[OF i] IH
    by (metis i id-take-nth-drop length-map nth-map)
qed simp
```

lemma *replace-at-fun-poss-not-below*:
assumes $\neg\ p \leq_p q$
and $p \in$ *poss t* **and** $q \in$ *fun-poss* (*replace-at t p s*)
shows $q \in$ *fun-poss t*
using *assms*
by (*metis ctxt-supt-id fun-poss-ctxt-apply-term hole-pos-ctxt-of-pos-term less-eq-pos-simps*(*1*))

lemma *substitution-subterm-at*:
assumes $\forall j <$ *length* (*vars-term-list l*). σ (*vars-term-list l* ! *j*) = *s* |- (*var-poss-list l* ! *j*)
and $\exists \tau.\ l \cdot \tau = s$
shows $l \cdot \sigma = s$
using *assms* **proof**(*induct l arbitrary*:*s*)
case (*Var x*)
then show *?case*
unfolding *vars-term-list.simps poss-list.simps var-poss.simps eval-term.simps*
by *simp*
next
case (*Fun f ts*)
from *Fun*(*3*) **obtain** *ss* **where** *s*:*s* = *Fun f ss* **and** *l*:*length ts* = *length ss*
by *fastforce*
{fix *i* **assume** *i*:*i* < *length ts*
{fix *j* **assume** *j*:*j*<*length* (*vars-term-list* (*ts*!*i*))
let *?p*=*var-poss-list* (*ts*!*i*) ! *j*
let *?x*=*vars-term-list* (*ts*!*i*) ! *j*
let *?k*=*sum-list* (*map* (*length* ∘ *vars-term-list*) (*take i ts*)) + *j*
from *i j* **have** *x*:*?x* = *vars-term-list* (*Fun f ts*) ! *?k*
unfolding *vars-term-list.simps* **by** (*simp add*: *concat-nth take-map*)
have *p*:*var-poss-list* (*Fun f ts*) ! *?k* = *i* # *?p* **proof**−
from *i* **have** *i'*:*i* < *length* (*map2* (λ*x*. *map* ((#) *x*)) [*0*..<*length ts*] (*map var-poss-list ts*))
by *simp*
from *i j* **have** *j* < *length* ((*map var-poss-list ts*) ! *i*)
using *length-var-poss-list* **by** (*metis* (*mono-tags*, *lifting*) *nth-map*)
with *i* **have** *j'*:*j* < *length* (*map2* (λ*x*. *map* ((#) *x*)) [*0*..<*length ts*] (*map var-poss-list ts*) ! *i*)
by *simp*
{fix *l* **assume** *l* < *length ts*
then have (*map length* (*map2* (λ*x*. *map* ((#) *x*)) [*0*..<*length ts*] (*map var-poss-list ts*)))!*l* = (*map* (*length* ∘ *vars-term-list*) *ts*) ! *l*
using *length-var-poss-list* **by** *simp*
}
then have *map length* (*map2* (λ*x*. *map* ((#) *x*)) [*0*..<*length ts*] (*map var-poss-list ts*)) = *map* (*length* ∘ *vars-term-list*) *ts*
using *nth-equalityI*[**where** *ys*=*map* (*length* ∘ *vars-term-list*) *ts*] **by** *simp*
with *i* **have** *k*:*sum-list* (*map length* (*take i* (*map2* (λ*x*. *map* ((#) *x*)) [*0*..<*length ts*] (*map var-poss-list ts*)))) + *j* = *?k*
by (*metis take-map*)

```
    then have var-poss-list (Fun f ts) ! ?k = (map2 (λi. map ((#) i)) [0..<length
ts] (map var-poss-list ts))!i !j
      unfolding var-poss-list.simps using concat-nth[OF i' j'] by presburger
     also have ... = (map ((#) i) (var-poss-list (ts!i)))!j using i by simp
     also have ... = i # ?p using nth-map j length-var-poss-list by metis
     ultimately show ?thesis by simp
    qed
    from i j have k:?k < length (vars-term-list (Fun f ts))
      unfolding vars-term-list.simps by (metis concat-nth-length length-map
map-map nth-map take-map)
    from Fun(2) k have σ ?x = (ss!i) |- (var-poss-list (ts!i) ! j)
     unfolding x s using p by simp
   }
   then have ∀j<length (vars-term-list (ts!i)).σ (vars-term-list (ts!i) ! j) = (ss!i)
|- var-poss-list (ts!i) ! j
    by simp
   moreover from Fun(3) have ∃τ. (ts!i) · τ = ss!i
    unfolding eval-term.simps s using i l by (metis nth-map term.inject(2))
   ultimately have (ts!i) · σ = ss!i
    using i Fun(1) nth-mem by blast
  }
  then show ?case unfolding eval-term.simps s
   using l by (simp add: map-nth-eq-conv)
qed

lemma vars-map-vars-term:
  map f (vars-term-list t) = vars-term-list (map-vars-term f t)
unfolding map-vars-term-eq proof(induct t)
  case (Fun g ts)
  then have map (λxs. map f xs)(map vars-term-list ts) = map vars-term-list (map
(λt. t · (Var ∘ f)) ts)
   by fastforce
 then show ?case unfolding vars-term-list.simps eval-term.simps map-map map-concat
   by presburger
qed (simp add: vars-term-list.simps)

lemma ctxt-apply-subt-at:
  assumes q ∈ poss s
  shows (ctxt-of-pos-term p (s|-q)) ⟨t⟩ = (ctxt-of-pos-term (q@p) s) ⟨t⟩ |- q
using assms proof(induct q arbitrary: s)
  case (Cons i q)
  from Cons(2) obtain f ss where i:i < length ss and s:s = Fun f ss
   by (meson args-poss)
  from i Cons show ?case unfolding s
   by (metis ctxt-apply-ctxt-apply ctxt-supt-id replace-at-subt-at)
qed simp
```

1.3.1 Utilities for *mk-subst*

We consider the special case of applying *mk-subst* when the variables involved form a partition.

lemma *mk-subst-same*:
assumes *length xs = length ts distinct xs*
shows *map (mk-subst f (zip xs ts)) xs = ts*
using *assms* **by** (*simp add*: *mk-subst-distinct map-nth-eq-conv*)

lemma *map2-zip*: *set (map fst (concat (map2 zip xs ys))) ⊆ set (concat xs)*
proof
fix *x* **assume** *x*:*x ∈ set (map fst (concat (map2 zip xs ys)))*
let *?l*=*min (length xs) (length ys)*
from *x* **obtain** *i* **where** *i*:*i < ?l x ∈ set (map fst (zip (xs!i) (ys!i)))*
by (*smt* (*verit*) *case-prod-conv in-set-conv-nth length-map length-zip min.strict-boundedE nth-concat-split nth-map nth-zip*)
then have *x ∈ set (xs!i)*
by (*metis in-set-takeD map-fst-zip-take*)
then show *x ∈ set (concat xs)*
using *i*(*1*) **by** (*metis concat-nth concat-nth-length in-set-conv-nth min.strict-boundedE*)

qed

lemma *mk-subst-partition*:
fixes *xs* :: *'a list list*
assumes *l*:*length xs = length ss*
and *part*:*is-partition (map set xs)*
shows *∀ i < length xs. ∀ x ∈ set (xs!i). (mk-subst f (zip (xs!i) (ss!i))) x = (mk-subst f (concat (map2 zip xs ss))) x*
proof−
{fix *i* **assume** *i*:*i < length xs*
{fix *x* **assume** *x*:*x ∈ set (xs!i)*
have *concat (map2 zip xs ss) = concat (map2 zip (take i xs) (take i ss)) @ concat (map2 zip (drop i xs) (drop i ss))*
by (*metis append-take-drop-id concat-append drop-map drop-zip take-map take-zip*)
moreover have *concat (map2 zip (drop i xs) (drop i ss)) = concat (zip (xs!i) (ss!i) # (map2 zip (drop (Suc i) xs) (drop (Suc i) ss)))*
using *i l* **by** (*smt* (*verit*, *del-insts*) *Cons-nth-drop-Suc list.map*(*2*) *prod.simps*(*2*) *zip-Cons-Cons*)
ultimately have *cc*:*concat (map2 zip xs ss) = concat (map2 zip (take i xs) (take i ss)) @*
concat (zip (xs!i) (ss!i) # (map2 zip (drop (Suc i) xs) (drop (Suc i) ss))) **by** *presburger*
{fix *j* **assume** *j < length xs* **and** *j ≠ i*
with *i x part* **have** *x ∉ set (xs!j)*
unfolding *is-partition-alt is-partition-alt-def* **by** *auto*
} note *part*=*this*
then have *x ∉ set (concat (take i xs))*

by (*smt* (*verit*) *in-set-conv-nth length-take less-length-concat min.strict-boundedE nth-take order-less-irrefl*)
then have $x \notin$ *set* (*map fst* (*concat* (*map2 zip* (*take i xs*) (*take i ss*))))
using *map2-zip in-mono* **by** *fastforce*
then have *subst*:(*mk-subst f* (*concat* (*map2 zip xs ss*))) x = (*mk-subst f* (*concat* (*zip* (*xs*!*i*) (*ss*!*i*) # (*map2 zip* (*drop* (*Suc i*) *xs*) (*drop* (*Suc i*) *ss*))))) x
unfolding *cc* **using** *mk-subst-concat* **by** *metis*
have *mk-subst f* (*zip* (*xs* ! *i*) (*ss* ! *i*)) x = (*mk-subst f* (*concat* (*map2 zip xs ss*))) x
proof(*cases* $x \in$ *set* (*map fst* (*zip* (*xs*!*i*) (*ss*!*i*))))
case *True*
then show *?thesis*
using *mk-subst-concat-Cons subst* **by** *metis*
next
case *False*
{fix *j* **assume** *j*:$j <$ *length* (*drop* (*Suc i*) *xs*)
then have (*drop* (*Suc i*) *xs*)!j = *xs*!(*Suc i* + j)
using *Suc-leI i nth-drop* **by** *blast*
moreover from *i j* **have** *Suc i* + $j <$ *length xs*
by (*metis add.commute length-drop less-diff-conv*)
ultimately have $x \notin$ *set* ((*drop* (*Suc i*) *xs*)!j)
using *part* **by** (*metis Suc-n-not-le-n le-add1*)
}
then have $x \notin$ *set* (*concat* (*drop* (*Suc i*) *xs*))
by (*smt* (*verit*) *in-set-conv-nth length-map length-take less-not-refl2 min.strict-boundedE nth-concat-split nth-map nth-take*)
then have $x \notin$ *set* (*map fst* (*concat* (*map2 zip* (*drop* (*Suc i*) *xs*) (*drop* (*Suc i*) *ss*))))
using *map2-zip in-mono* **by** *fastforce*
with *False* **have** $x \notin$ *set* (*map fst* (*concat* (*zip* (*xs*!*i*) (*ss*!*i*) # (*map2 zip* (*drop* (*Suc i*) *xs*) (*drop* (*Suc i*) *ss*)))))
unfolding *concat.simps* **by** (*metis Un-iff map-append set-append*)
with *False* **show** *?thesis*
unfolding *subst* **using** *mk-subst-not-mem*′ **by** *metis*
qed
}
}
then show *?thesis* **by** *simp*
qed

The following lemma is used later to show that $A = (\textit{to-pterm}\ (\textit{lhs}\ \alpha)) \cdot \sigma$ implies $A = (\textit{to-pterm}\ (\textit{lhs}\ \alpha)) \cdot \langle \textit{As} \rangle_\alpha$ for some suitable *As*.

lemma *subst-imp-mk-subst*:
assumes $s = t \cdot \sigma$
shows $\exists$ *ss*. $t \cdot \sigma = t \cdot$ (*mk-subst Var* (*zip* (*vars-distinct t*) *ss*)) $\wedge$ *length ss* = *length* (*vars-distinct t*) $\wedge$ ($\forall i <$ *length ss*. σ (*vars-distinct t*!i) = *ss*!i)
proof−

```
  let ?ss=map σ (vars-distinct t)
  let ?τ=(mk-subst Var (zip (vars-distinct t) ?ss))
  {fix x assume x ∈ vars-term t
    then have σ x = ?τ x unfolding mk-subst-def
      by (simp add: map-of-zip-map)
  }
  then have t · σ = t · ?τ
    using term-subst-eq by blast
  then show ?thesis by auto
qed

lemma mk-subst-rename:
  assumes length (vars-distinct t) = length xs and inj f
   shows t · (mk-subst Var (zip (vars-distinct t) xs)) = (map-vars-term f t) ·
(mk-subst Var (zip (vars-distinct (map-vars-term f t)) xs))
proof−
  {fix x assume x ∈ vars-term t
    then obtain i where i:x = (vars-distinct t)!i i < length (vars-distinct t)
      by (metis in-set-conv-nth set-vars-term-list vars-term-list-vars-distinct)
    with assms have 1:(mk-subst Var (zip (vars-distinct t) xs)) x = xs!i
      using mk-subst-distinct by (metis comp-apply distinct-remdups distinct-rev)
    have vars-distinct (map-vars-term f t) = map f (vars-distinct t)
      unfolding vars-map-vars-term[symmetric] comp-apply using assms(2)
       by (metis distinct-map distinct-remdups distinct-remdups-id inj-on-inverseI
remdups-map-remdups rev-map the-inv-f-f)
     with assms i have 2:(mk-subst Var (zip (vars-distinct (map-vars-term f t))
xs)) (f x) = xs!i
     by (metis (mono-tags, lifting) comp-apply distinct-remdups distinct-rev length-map
mk-subst-same nth-map)
   from 1 2 have (mk-subst Var (zip (vars-distinct t) xs)) x = (mk-subst Var (zip
(vars-distinct (map-vars-term f t)) xs)) (f x)
      by presburger
  }
  then show ?thesis
    by (simp add: apply-subst-map-vars-term term-subst-eq-conv)
qed
```

1.4 Matching Terms

The goal is showing that *match* $(t \cdot \sigma)$ t = *Some* σ whenever the domain of σ is a subset of the variables in t. For that we need some helper lemmas.

```
lemma decompose-fst:
  assumes decompose (Fun f ss) t = Some us
  shows map fst us = ss
proof−
  from assms obtain ts where t:t = Fun f ts
     by (metis (no-types, lifting) decompose-def option.distinct(1) decompose-Some
is-FunE old.prod.case term.case-eq-if)
  with assms have length ss = length ts
```

by *blast*
with *assms*(*1*) *t* **show** *?thesis*
by *auto*
qed

lemma *decompose-vars-term*:
assumes *decompose* (*Fun f ss*) *t* = *Some us*
shows *vars-term* (*Fun f ss*) = (⋃(*a*, *b*) ∈ *set us*. *vars-term a*)
proof−
have *vars-term* (*Fun f ss*) = (⋃ *s* ∈ *set ss*. *vars-term s*)
by (*meson Term.term.simps*(*18*))
also have ... = (⋃ *s* ∈ *set* (*map fst us*). *vars-term s*)
using *assms decompose-fst* **by** *metis*
finally show *?thesis*
using *image-image* **by** *auto*
qed

lemma *match-term-list-domain*:
assumes *match-term-list P σ* = *Some τ*
shows ∀ *x*. *x* ∉ (⋃ (*a*, *b*) ∈ *set P*. *vars-term a*) ∧ *σ x* = *None* ⟶ *τ x* = *None*
using *assms* **proof**(*induct P σ rule*:*match-term-list.induct*)
case (*2 x t P σ*)
then show *?case*
by (*metis* (*mono-tags*, *lifting*) *Sup-insert Un-iff case-prod-conv fun-upd-idem-iff fun-upd-triv fun-upd-twist image-insert list.simps*(*15*) *match-term-list.simps*(*2*) *option.simps*(*3*) *term.set-intros*(*3*))
next
case (*3 f ss g ts P σ*)
from *3*(*2*) **obtain** *us* **where** *us*:*decompose* (*Fun f ss*) (*Fun g ts*) = *Some us*
using *match-term-list.simps*(*3*) *option.distinct*(*1*) *option.simps*(*4*) **by** *fastforce*
with *3*(*2*) **have** ∗:*match-term-list* (*us* @ *P*) *σ* = *Some τ*
by *auto*
from *us* **have** (⋃ (*a*, *b*) ∈ *set* ((*Fun f ss*, *Fun g ts*) # *P*). *vars-term a*) = (⋃ *s* ∈ *set ss*. *vars-term s*) ∪ (⋃ (*a*, *b*) ∈ *set P*. *vars-term a*)
by *simp*
also have ... = (⋃ (*a*, *b*) ∈ *set* (*us*@*P*). *vars-term a*)
using *us* **by** (*metis* (*mono-tags*, *lifting*) *Term.term.simps*(*18*) *UN-Un decompose-vars-term set-append*)
finally show *?case*
using *3*(*1*) *us* ∗ **by** *metis*
qed *simp-all*

lemma *match-subst-domain*:
assumes *match a b* = *Some σ*
shows *subst-domain σ* ⊆ *vars-term b*
proof−
from *assms* **have** ∀ *x*. *x* ∉ *vars-term b* ⟶ *σ x* = *Var x*
unfolding *match-def match-list-def subst-of-map-def* **using** *match-term-list-domain* **by** *fastforce*

then show *?thesis*
using *subst-domain-def* **by** *fastforce*
qed

lemma *match-trivial*:
assumes *subst-domain* $\sigma \subseteq$ *vars-term t*
shows *match* $(t \cdot \sigma)$ t = *Some* σ
proof−
obtain τ **where** *tau*:*match* $(t \cdot \sigma)$ t = *Some* τ **and** *1*:$(\forall x \in$*vars-term t*. $\sigma\ x = \tau\ x)$
by (*meson match-complete'*)
from *assms* **have** *2*:$\forall x.\ x \notin$ *vars-term t* $\longrightarrow \sigma\ x =$ *Var x*
by (*meson notin-subst-domain-imp-Var subset-eq*)
from *tau* **have** *3*:$\forall x.\ x \notin$ *vars-term t* $\longrightarrow \tau\ x =$ *Var x*
using *match-subst-domain notin-subst-domain-imp-Var* **by** *fastforce*
from *1 2 3* **show** *?thesis*
by (*metis subst-term-eqI tau term-subst-eq*)
qed

end

1.4.1 Matching of Linear Terms

theory *Linear-Matching*
imports *Proof-Term-Utils*
begin

For a linear term the matching substitution can simply be computed with the following definition.

definition *match-substs* :: ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *term* ⇒ ('*v* × ('*f*, '*v*) *term*) *list*
where *match-substs t s* = (*zip* (*vars-term-list t*) (*map* (λ*p*. *s*|-*p*) (*var-poss-list t*)))

lemma *mk-subst-partition-special*:
assumes *length ss* = *length ts*
and *is-partition* (*map vars-term ts*)
shows ∀ *i* < *length ts*. (*ts*!*i*) · (*mk-subst f* (*zip* (*vars-term-list* (*ts*!*i*)) (*ss*!*i*))) = (*ts*!*i*) · (*mk-subst f* (*concat* (*map2 zip* (*map vars-term-list ts*) *ss*)))
proof−
let *?xs*=*map vars-term-list ts*
have *xs*:*map vars-term ts* = *map set* (*map vars-term-list ts*) **by** *simp*
from *assms*(*1*) **have** *l*:*length ?xs* = *length ss* **by** *simp*
{fix *i* **assume** *i*:*i* < *length ts*
{fix *x* **assume** *x* ∈ *vars-term* (*ts*!*i*)
then have *mk-subst f* (*zip* (*map vars-term-list ts* ! *i*) (*ss* ! *i*)) *x* = *mk-subst f* (*concat* (*map2 zip* (*map vars-term-list ts*) *ss*)) *x*
using *i mk-subst-partition*[*OF l assms*(*2*)[*unfolded xs*]] **by** *simp*
}
then have *ts*!*i* · (*mk-subst f* (*zip* (*vars-term-list* (*ts*!*i*)) (*ss*!*i*))) = (*ts*!*i*) ·

```
(mk-subst f (concat (map2 zip (map vars-term-list ts) ss)))
      by (simp add: i term-subst-eq-conv)
  }
  then show ?thesis by fastforce
qed

lemma match-substs-Fun:
  assumes l:length ts = length ss
  shows match-substs (Fun f ts) (Fun g ss) = concat (map2 zip (map vars-term-list
ts) (map2 (λt s. map ((|-) s) (var-poss-list t)) ts ss))
    (is match-substs (Fun f ts) (Fun g ss) = concat (map2 zip ?xs ?terms))
proof−
  have l′:length ?xs = length ?terms using l by simp
  {fix i assume i < length ?xs
    then have i:i < length ts by simp
    with l have zip:(zip ts ss)!i = (ts!i, ss!i) by simp
    from i l have length (map vars-term-list ts ! i) = length (map (λp. (ss!i)|-p)
(var-poss-list (ts!i)))
      by (simp add: length-var-poss-list)
    with zip have length (?xs!i) = length (?terms!i)
      using i l′ by auto
  }note l-i=this
  have vars-term-list (Fun f ts) = concat ?xs
    unfolding vars-term-list.simps by simp
  moreover have map ((|-) (Fun g ss)) (var-poss-list (Fun f ts)) = concat ?terms
proof−
    have l-map2:length (map2 (λi. map ((#) i)) [0..<length ts] (map var-poss-list
ts)) = length ts
      unfolding length-map length-zip by simp
    {fix i assume i:i < length ts
     with l have length (map2 (λi. map ((#) i)) [0..<length ts] (map var-poss-list
ts) !i) = length (map var-poss-list ts!i)
       unfolding nth-map[OF i] by simp
    }
    with l-map2 have length (map ((|-) (Fun g ss)) (var-poss-list (Fun f ts))) =
length (concat (map var-poss-list ts))
    unfolding var-poss-list.simps length-map length-concat by (smt (verit, del-insts)
length-map map-nth-eq-conv)
   moreover have length (concat ?terms) = length (concat (map var-poss-list ts))
proof−
     {fix i assume i < length ts
       with l have length (map2 (λt s. map ((|-) s) (var-poss-list t)) ts ss ! i) =
length (map var-poss-list ts!i) by simp
     }
     moreover have length (map2 (λt s. map ((|-) s) (var-poss-list t)) ts ss) =
length ts using l by simp
    ultimately show ?thesis unfolding length-concat  by (smt (verit, del-insts)
length-map map-nth-eq-conv)
   qed
```

ultimately have *l''*:*length* (*map* ((|-) (*Fun g ss*)) (*var-poss-list* (*Fun f ts*))) = *length* (*concat ?terms*) **by** *presburger*
{fix *i* **assume** *i*:*i* < *length* (*var-poss-list* (*Fun f ts*))
let *?ps*=*map2* (λ*i*. *map* ((#) *i*)) [*0*..<*length ts*] (*map var-poss-list ts*)
let *?p*=*var-poss-list* (*Fun f ts*) ! *i*
from *l* **have** *l-terms*:*length ?terms* = *length ts* **by** *auto*
from *l* **have** *l-ps*:*length ?ps* = *length ts* **by** *auto*
obtain *j k* **where** *j*:*j* < *length ts* **and** *k*:*k* < *length* (*var-poss-list* (*ts*!*j*)) **and** *i-sum-list*:*i* = *sum-list* (*map length* (*take j ?ps*)) + *k*
and ∗:*?p* = *map2* (λ*i*. *map* ((#) *i*)) [*0*..<*length ts*] (*map var-poss-list ts*) ! *j* ! *k*
using *less-length-concat*[*OF i*[*unfolded var-poss-list.simps*]] **by** *auto*
let *?p'*=(*var-poss-list* (*ts*!*j*)) ! *k*
from ∗ *k j* **have** *p*:*?p* = *j* # *?p'* **by** *simp*
from *j l* **have** *1*:(*Fun g ss*) |- *?p* = (*ss*!*j*) |- *?p'* **unfolding** *p* **by** *simp*
have *i-sum-list2*:*i* = *sum-list* (*map length* (*take j ?terms*)) + *k* **proof**−
{fix *n* **assume** *n* < *length ts*
with *l* **have** *length* (*?terms*!*n*) = *length* (*?ps*!*n*) **by** *auto*
}
then have *map length ?terms* = *map length ?ps*
using *l-terms l-ps* **by** (*simp add*: *map-eq-conv'*)
then show *?thesis* **unfolding** *i-sum-list* **by** (*metis take-map*)
qed
from *j k* **have** *k* < *length* (*?terms* ! *j*) **by** (*smt* (*verit*) *l-i length-map length-var-poss-list nth-map*)
with *j i-sum-list2* **have** *concat ?terms* ! *i* = *?terms* ! *j* ! *k*
using *concat-nth*[*of j ?terms k i*] **unfolding** *length-map length-zip l min.idem* **by** *auto*
then have *2*:*concat ?terms* ! *i* = (*ss*!*j*) |- *?p'* **using** *k j l* **by** *auto*
from *1 2* **have** *map* ((|-) (*Fun g ss*)) (*var-poss-list* (*Fun f ts*)) ! *i* = *concat ?terms* ! *i*
unfolding *var-poss-list.simps nth-map*[*OF i*[*unfolded var-poss-list.simps*]] **by** *simp*
}
with *l''* **show** *?thesis* **by** (*metis length-map nth-equalityI*)
qed
ultimately show *?thesis*
unfolding *match-substs-def* **using** *concat-map2-zip*[*OF l'*] *l-i* **by** *presburger*
qed

If all function symbols in term t coincide with function symbols in term s, then t matches s.

lemma *fun-poss-eq-imp-matches*:
fixes *s t* :: ('*f*, '*v*) *term*
assumes *linear-term t* **and** ∀ *p* ∈ *poss t*. ∀ *f ts*. *t*|-*p* = *Fun f ts* ⟶ (∃ *ss*. *length ts* = *length ss* ∧ *s*|-*p* = *Fun f ss*)
shows *t* · (*mk-subst Var* (*match-substs t s*)) = *s*
using *assms* **proof**(*induct t arbitrary*:*s*)
case (*Var x*)

```
  have match-substs (Var x) s = [(x, s)]
    unfolding match-substs-def var-poss-list.simps vars-term-list.simps by simp
  then show ?case by simp
next
  case (Fun f ts)
  from Fun(3) obtain ss where l:length ts = length ss and s:s = Fun f ss by
auto
  let ?σ=mk-subst Var (match-substs (Fun f ts) (Fun f ss))
  let ?xs=map vars-term-list ts
  let ?ss=map (λ(t, s). map (λp. s|-p) (var-poss-list t)) (zip ts ss)
  have concat-zip:match-substs (Fun f ts) (Fun f ss) = concat (map2 zip ?xs ?ss)
    unfolding match-substs-Fun[OF l] by simp
  from Fun(2) have part:is-partition (map set ?xs)
     by (smt (verit, ccfv-SIG) length-map linear-term.elims(2) map-nth-eq-conv
set-vars-term-list term.distinct(1) term.sel(4))
  have l':length ?xs = length ?ss using l by simp
  {fix i assume i:i < length ts
    with Fun(2) have lin:linear-term (ts!i) by simp
    let ?σi=mk-subst Var (match-substs (ts!i) (ss!i))
    {fix p f' ts' assume p:p ∈ poss (ts!i) ts!i |- p = Fun f' ts'
      from p(1) i have i#p ∈ poss (Fun f ts) by simp
      moreover from p(2) i have (Fun f ts)|-(i#p) = Fun f' ts' by simp
      ultimately obtain ss' where length ts' = length ss' and s|-(i#p) = Fun f'
ss' using Fun(3) by blast
      then have ∃ ss'. length ts' = length ss' ∧ (ss!i)|-p = Fun f' ss' unfolding s
by simp
    }
    then have ∀ p ∈ poss (ts!i). ∀ f' ts'. (ts!i)|-p = Fun f' ts' ⟶ (∃ ss'. length ts'
= length ss' ∧ (ss!i)|-p = Fun f' ss') by simp
    with Fun(1) lin i have IH:(ts!i) · ?σi = ss!i using nth-mem by blast
    have (ts!i) · ?σ = (ts!i) · ?σi proof−
      {fix x assume x:x ∈ vars-term (ts!i)
        from i l have ∗:map2 (λt s. map ((|-) s) (var-poss-list t)) ts ss ! i = map
((|-) (ss ! i)) (var-poss-list (ts ! i)) by auto
        with i x have ?σ x = ?σi x
          unfolding concat-zip using mk-subst-partition[OF l' part] unfolding s
match-substs-Fun[OF l] match-substs-def length-map
          by (smt (verit, best) nth-map set-vars-term-list)
      }
      then show ?thesis by (simp add: term-subst-eq-conv)
    qed
    then have (ts!i) · ?σ = ss!i using IH i by presburger
  }
  then show ?case unfolding s by (simp add: l map-nth-eq-conv)
qed

end
```

2 Proof Terms

theory *Proof-Terms*
imports
First-Order-Terms.Matching
First-Order-Rewriting.Multistep
Proof-Term-Utils
begin

2.1 Definitions

A rewrite rule consists of a pair of terms representing its left-hand side and right-hand side. We associate a rule symbol with each rewrite rule.

datatype $('f, 'v)$ *prule* =
Rule (*lhs*: $('f, 'v)$ *term*) (*rhs*: $('f, 'v)$ *term*) (- → - [*51*, *51*] *52*)

Translate between *prule* defined here and *rule* as defined in IsaFoR.

abbreviation *to-rule* :: $('f, 'v)$ *prule* ⇒ $('f, 'v)$ *rule*
where *to-rule r* ≡ (*lhs r*, *rhs r*)

Proof terms are terms built from variables, function symbols, and rules.

type-synonym
$('f, 'v)$ *pterm* = $(('f, 'v)$ *prule* + $'f, 'v)$ *term*
type-synonym
$('f, 'v)$ *pterm-ctxt* = $(('f, 'v)$ *prule* + $'f, 'v)$ *ctxt*

We provides an easier notation for proof terms (avoiding *Inl* and *Inr*).

abbreviation *Prule* :: $('f, 'v)$ *prule* ⇒ $('f, 'v)$ *pterm list* ⇒ $('f, 'v)$ *pterm*
where *Prule* α *As* ≡ *Fun* (*Inl* α) *As*
abbreviation *Pfun* :: $'f$ ⇒ $('f, 'v)$ *pterm list* ⇒ $('f, 'v)$ *pterm*
where *Pfun f As* ≡ *Fun* (*Inr f*) *As*

Also for contexts.

abbreviation *Crule* :: $('f, 'v)$ *prule* ⇒ $('f, 'v)$ *pterm list* ⇒ $('f, 'v)$ *pterm-ctxt* ⇒ $('f, 'v)$ *pterm list* ⇒$('f, 'v)$ *pterm-ctxt*
where *Crule* α *As1 C As2* ≡ *More* (*Inl* α) *As1 C As2*
abbreviation *Cfun* :: $'f$ ⇒ $('f, 'v)$ *pterm list* ⇒ $('f, 'v)$ *pterm-ctxt* ⇒ $('f, 'v)$ *pterm list* ⇒$('f, 'v)$ *pterm-ctxt*
where *Cfun f As1 C As2* ≡ *More* (*Inr f*) *As1 C As2*

Case analysis on proof terms.

lemma *pterm-cases* [*case-names Var Pfun Prule*, *cases type*: *pterm*]:
(⋀*x*. *A* = *Var x* ⟹ *P*) ⟹ (⋀*f As*. *A* = *Pfun f As* ⟹ *P*) ⟹ (⋀α *As*. *A* = *Prule* α *As* ⟹ *P*) ⟹ *P*
proof (*cases A*)
case (*Fun x21 x22*)
show ⋀*x21 x22*. (⋀*x*. *A* = *Var x* ⟹ *P*) ⟹ (⋀*f As*. *A* = *Pfun f As* ⟹ *P*) ⟹ (⋀α *As*. *A* = *Prule* α *As* ⟹ *P*) ⟹ *A* = *Fun x21 x22* ⟹ *P*

using *sum.exhaust* **by** *auto*
qed

Induction scheme for proof terms.

lemma
fixes $P :: ('f, 'v)$ *pterm* $\Rightarrow$ *bool*
assumes $\bigwedge x.$ P (*Var* x)
and $\bigwedge f$ *As*. $(\bigwedge a.$ $a \in$ *set As* $\Longrightarrow$ P $a) \Longrightarrow$ P (*Pfun f As*)
and $\bigwedge \alpha$ *As*. $(\bigwedge a.$ $a \in$ *set As* $\Longrightarrow$ P $a) \Longrightarrow$ P (*Prule* α *As*)
shows *pterm-induct* [*case-names Var Pfun Prule*, *induct type*: *pterm*]: P *A*
using *assms* **proof**(*induct A*)
case (*Fun f ts*)
then show *?case* **by**(*cases f*) *auto*
qed *simp*

Induction scheme for contexts of proof terms.

lemma
fixes $P :: ('f, 'v)$ *pterm-ctxt* $\Rightarrow$ *bool*
assumes P $\Box$
and $\bigwedge f$ *ss1 C ss2*. P *C* $\Longrightarrow$ P (*Cfun f ss1 C ss2*)
and $\bigwedge \alpha$ *ss1 C ss2*. P *C* $\Longrightarrow$ P (*Crule* α *ss1 C ss2*)
shows *pterm-ctxt-induct* [*case-names Hole Cfun Crule*, *induct type*: *pterm-ctxt*]: P *C*
using *assms* **proof**(*induct C*)
case (*More f ss1 C ss2*)
then show *?case* **by**(*cases f*) *auto*
qed

Obtain the distinct variables occurring on the left-hand side of a rule in the order they appear.

abbreviation *var-rule* :: $('f, 'v)$ *prule* $\Rightarrow$ $'v$ *list* **where** *var-rule* $\alpha \equiv$ *vars-distinct* (*lhs* α)

abbreviation *lhs-subst* :: $('g, 'v)$ *term list* $\Rightarrow$ $('f, 'v)$ *prule* $\Rightarrow$ $('g, 'v)$ *subst* (⟨-⟩_ [*0*,*99*])
where *lhs-subst As* $\alpha \equiv$ *mk-subst Var* (*zip* (*var-rule* α) *As*)

A proof term using only function symbols and variables is an empty step.

fun *is-empty-step* :: $('f, 'v)$ *pterm* $\Rightarrow$ *bool* **where**
is-empty-step (*Var x*) = *True*
| *is-empty-step* (*Pfun f As*) = *list-all is-empty-step As*
| *is-empty-step* (*Prule* α *As*) = *False*

fun *is-Prule* :: $('f, 'v)$ *pterm* $\Rightarrow$ *bool* **where**
is-Prule (*Prule* - -) = *True*
| *is-Prule* - = *False*

Source and target

fun *source* :: $('f, 'v)$ *pterm* $\Rightarrow$ $('f, 'v)$ *term* **where**
  source (*Var x*) = *Var x*
| *source* (*Pfun f As*) = *Fun f* (*map source As*)
| *source* (*Prule* α *As*) = *lhs* $\alpha \cdot \langle$*map source As*$\rangle_\alpha$

fun *target* :: $('f, 'v)$ *pterm* $\Rightarrow$ $('f, 'v)$ *term* **where**
  target (*Var x*) = *Var x*
| *target* (*Pfun f As*) = *Fun f* (*map target As*)
| *target* (*Prule* α *As*) = *rhs* $\alpha \cdot \langle$*map target As*$\rangle_\alpha$

Source also works for proof term contexts in left-linear TRSs.

fun *source-ctxt* :: $('f, 'v)$ *pterm-ctxt* $\Rightarrow$ $('f, 'v)$ *ctxt* **where**
  source-ctxt $\Box$ = $\Box$
| *source-ctxt* (*Cfun f As1 C As2*) = *More f* (*map source As1*) (*source-ctxt C*) (*map source As2*)
| *source-ctxt* (*Crule* α *As1 C As2*) =
  (*let ctxt-pos* = (*var-poss-list* (*lhs* α))!(*length As1*);
      placeholder = *Var* ((*vars-term-list* (*lhs* α)) ! (*length As1*)) *in*
  ctxt-of-pos-term (*ctxt-pos*) (*lhs* $\alpha \cdot \langle$*map source* (*As1* @ ((*placeholder* # *As2*)))$\rangle_\alpha$))
$\circ_c$ (*source-ctxt C*)

abbreviation *co-initial A B* $\equiv$ (*source A* = *source B*)

Transform simple terms to proof terms.

fun *to-pterm* :: $('f, 'v)$ *term* $\Rightarrow$ $('f, 'v)$ *pterm* **where**
  to-pterm (*Var x*) = *Var x*
| *to-pterm* (*Fun f ts*) = *Pfun f* (*map to-pterm ts*)

Also for contexts.

fun *to-pterm-ctxt* :: $('f, 'v)$ *ctxt* $\Rightarrow$ $('f, 'v)$ *pterm-ctxt* **where**
  to-pterm-ctxt $\Box$ = $\Box$
| *to-pterm-ctxt* (*More f ss1 C ss2*) = *Cfun f* (*map to-pterm ss1*) (*to-pterm-ctxt C*) (*map to-pterm ss2*)

2.2 Frequently Used Locales/Contexts

Often certain properties about proof terms only hold when the underlying TRS does not contain variable left-hand sides and/or variables on the right are a subset of the variables on the left and/or the TRS is left-linear.

locale *left-lin* =
  fixes *R* :: $('f, 'v)$ *trs*
  assumes *left-lin*:*left-linear-trs R*

locale *no-var-lhs* =
  fixes *R* :: $('f, 'v)$ *trs*
  assumes *no-var-lhs*:*Ball R* (λ(*l*, *r*). *is-Fun l*)

locale *var-rhs-subset-lhs* =

fixes $R :: ('f, 'v)\ trs$
assumes *varcond*:*Ball* R $(\lambda(l, r).\ vars\text{-}term\ r \subseteq vars\text{-}term\ l)$

locale *wf-trs* = *no-var-lhs* + *var-rhs-subset-lhs*
locale *left-lin-no-var-lhs* = *left-lin* + *no-var-lhs*
locale *left-lin-wf-trs* = *left-lin* + *wf-trs*

context *wf-trs*
begin
lemma *wf-trs-alt*:
shows *Trs.wf-trs R*
unfolding *wf-trs-def'* **using** *no-var-lhs varcond* **by** *auto*
end

context *left-lin*
begin
lemma *length-var-rule*:
assumes *to-rule* $\alpha \in R$
shows *length* (*var-rule* α) = *length* (*vars-term-list* (*lhs* α))
using *assms*
by (*metis case-prodD left-lin left-linear-trs-def linear-term-var-vars-term-list*)
end

2.3 Proof Term Predicates

The number of arguments of a well-defined proof term over a TRS R using a rule symbol α corresponds to the number of variables in *lhs* α. Also the rewrite rule for α must belong to the TRS R.

inductive-set *wf-pterm* $:: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ pterm\ set$
for R **where**
[*simp*]: *Var* $x \in$ *wf-pterm* R
|[*intro*]: $\forall\, t \in set\ ts.\ t \in$ *wf-pterm* $R \Longrightarrow$ *Pfun* $f\ ts \in$ *wf-pterm* R
|[*intro*]: (*lhs* α, *rhs* α) $\in R \Longrightarrow$ *length* As = *length* (*var-rule* α) $\Longrightarrow$
$\forall\, a \in set\ As.\ a \in$ *wf-pterm* $R \Longrightarrow$ *Prule* $\alpha\ As \in$ *wf-pterm* R

inductive-set *wf-pterm-ctxt* $:: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ pterm\text{-}ctxt\ set$
for R **where**
[*simp*]: $\Box \in$ *wf-pterm-ctxt* R
|[*intro*]: $\forall\, s \in (set\ ss1) \cup (set\ ss2).\ s \in$ *wf-pterm* $R \Longrightarrow C \in$ *wf-pterm-ctxt* $R \Longrightarrow$
Cfun f ss1 C ss2 $\in$ *wf-pterm-ctxt* R
|[*intro*]: (*lhs* α, *rhs* α) $\in R \Longrightarrow$ (*length ss1*) + (*length ss2*) + 1 = *length* (*var-rule* α) $\Longrightarrow$
$\forall\, s \in (set\ ss1) \cup (set\ ss2).\ s \in$ *wf-pterm* $R \Longrightarrow C \in$ *wf-pterm-ctxt* $R \Longrightarrow$
Crule α *ss1 C ss2* $\in$ *wf-pterm-ctxt* R

lemma *fun-well-arg*[*intro*]:
assumes *Fun f As* $\in$ *wf-pterm* R $a \in set\ As$
shows $a \in$ *wf-pterm* R
using *assms wf-pterm.cases* **by** *auto*

lemma *trs-well-ctxt-arg*[*intro*]:
assumes *More f ss1 C ss2* ∈ *wf-pterm-ctxt R s* ∈ (*set ss1*) ∪ (*set ss2*)
shows *s* ∈ *wf-pterm R*
using *assms wf-pterm-ctxt.cases* **by** *blast*

lemma *trs-well-ctxt-C*[*intro*]:
assumes *More f ss1 C ss2* ∈ *wf-pterm-ctxt R*
shows *C* ∈ *wf-pterm-ctxt R*
using *assms wf-pterm-ctxt.cases* **by** *auto*

context *no-var-lhs*
begin
lemma *lhs-is-Fun*:
assumes *Prule* α *Bs* ∈ *wf-pterm R*
shows *is-Fun* (*lhs* α)
by (*metis Inl-inject assms case-prodD is-FunI is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *is-VarI*
no-var-lhs.no-var-lhs no-var-lhs-axioms term.inject(*2*) *wf-pterm.simps*)
end

lemma *lhs-subst-var-well-def*:
assumes ∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
shows (⟨*As*⟩α) *x* ∈ *wf-pterm R*
proof (*cases map-of* (*zip* (*var-rule* α) *As*) *x*)
case *None*
then show *?thesis* **unfolding** *mk-subst-def* **by** *simp*
next
case (*Some a*)
then have *a* ∈ *set As*
by (*meson in-set-zipE map-of-SomeD*)
with *assms Some* **show** *?thesis*
unfolding *mk-subst-def* **using** *in-set-idx* **by** *force*
qed

lemma *lhs-subst-well-def*:
assumes ∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R B* ∈ *wf-pterm R*
shows *B* · (⟨*As*⟩α) ∈ *wf-pterm R*
using *assms* **proof**(*induction B arbitrary*: *As*)
case (*Var x*)
then show *?case* **using** *lhs-subst-var-well-def* **by** *simp*
next
case (*Pfun f Bs*)
from *Pfun*(*3*) **have** ∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm R*
by *blast*
with *Pfun* **show** *?case* **by** *fastforce*
next
case (*Prule* β *Bs*)
from *Prule*(*3*) **have** ∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm R*

by *blast*
moreover have *length* (*map* (λ*t*. *t* · ⟨*As*⟩$_\alpha$) *Bs*) = *length* (*var-rule* β)
using *Prule*(*3*) *wf-pterm.simps* **by** *fastforce*
moreover from *Prule*(*3*) **have** *to-rule* β ∈ *R*
using *Inl-inject sum.distinct*(*1*) *wf-pterm.cases* **by** *force*
ultimately show *?case* **unfolding** *eval-term.simps*(*2*) **using** *Prule*
by (*simp add*: *wf-pterm.intros*(*3*))
qed

lemma *subt-at-is-wf-pterm*:
assumes *p* ∈ *poss A* **and** *A* ∈ *wf-pterm R*
shows *A*|-*p* ∈ *wf-pterm R*
using *assms* **proof**(*induct p arbitrary*:*A*)
case (*Cons i p*)
then obtain *f As* **where** *a*:*A* = *Fun f As* **and** *i*:*i* < *length As* **and** *p*:*p* ∈ *poss* (*As*!*i*)
using *args-poss* **by** *blast*
with *Cons*(*3*) **have** *As*!*i* ∈ *wf-pterm R*
using *nth-mem* **by** *blast*
with *Cons.hyps p a* **show** *?case* **by** *simp*
qed *simp*

lemma *ctxt-of-pos-term-well*:
assumes *p* ∈ *poss A* **and** *A* ∈ *wf-pterm R*
shows *ctxt-of-pos-term p A* ∈ *wf-pterm-ctxt R*
using *assms* **proof**(*induct p arbitrary*:*A*)
case (*Cons i p*)
then obtain *fs As* **where** *a*:*A* = *Fun fs As* **and** *i*:*i* < *length As* **and** *p*:*p* ∈ *poss* (*As*!*i*)
using *args-poss* **by** *blast*
with *Cons*(*3*) **have** *as*:∀ *j* < *length As*. *As*!*j* ∈ *wf-pterm R*
using *nth-mem* **by** *blast*
with *Cons.hyps p i* **have** *IH*:*ctxt-of-pos-term p* (*As*!*i*) ∈ *wf-pterm-ctxt R*
by *blast*
then show *?case* **proof**(*cases fs*)
case (*Inl* α)
from *Cons*(*3*) **have** *to-rule* α ∈ *R* **unfolding** *a Inl*
using *wf-pterm.cases* **by** *auto*
moreover from *Cons*(*3*) *i* **have** *length* (*take i As*) + *length* (*drop* (*Suc i*) *As*) + *1* = *length* (*var-rule* α)
unfolding *a Inl* **using** *wf-pterm.cases* **by** *force*
ultimately show *?thesis*
unfolding *a ctxt-of-pos-term.simps Inl* **using** *as IH wf-pterm-ctxt.intros*(*3*)
by (*metis* (*no-types*, *opaque-lifting*) *UnE in-set-conv-nth in-set-dropD in-set-takeD*)
next
case (*Inr f*)
show *?thesis*
unfolding *a ctxt-of-pos-term.simps Inr* **using** *as IH wf-pterm-ctxt.intros*(*2*)
by (*metis Cons.prems*(*2*) *UnE a fun-well-arg in-set-dropD in-set-takeD*)

qed
qed *simp*

Every normal term is a well-defined proof term.

lemma *to-pterm-wf-pterm*[*simp*]: *to-pterm t* ∈ *wf-pterm R*
by (*induction t*) (*simp-all add*: *wf-pterm.intros*(*2*,*3*))

lemma *to-pterm-trs-ctxt*:
assumes *p* ∈ *poss* (*to-pterm s*)
shows *ctxt-of-pos-term p* (*to-pterm s*) ∈ *wf-pterm-ctxt R*
by (*simp add*: *assms ctxt-of-pos-term-well*)

lemma *to-pterm-ctxt-wf-pterm-ctxt*:
shows *to-pterm-ctxt C* ∈ *wf-pterm-ctxt R*
proof(*induct C*)
case (*More f xs C ys*)
then show *?case* **unfolding** *to-pterm-ctxt.simps*
by (*metis Un-iff fun-well-arg to-pterm.simps*(*2*) *to-pterm-wf-pterm wf-pterm-ctxt.intros*(*2*))
qed *simp*

lemma *ctxt-wf-pterm*:
assumes *A* ∈ *wf-pterm R* **and** *p* ∈ *poss A*
and *B* ∈ *wf-pterm R*
shows (*ctxt-of-pos-term p A*)⟨*B*⟩ ∈ *wf-pterm R*
using *assms* **proof**(*induct p arbitrary*:*A*)
case (*Cons i p*)
from *Cons*(*3*) **obtain** *f As* **where** *A*:*A* = *Fun f As i* < *length As p* ∈ *poss* (*As*!*i*)
using *args-poss* **by** *blast*
moreover with *Cons*(*2*) **have** *As*!*i* ∈ *wf-pterm R*
using *nth-mem* **by** *blast*
ultimately have *IH*:(*ctxt-of-pos-term p* (*As*!*i*))⟨*B*⟩ ∈ *wf-pterm R*
using *Cons.hyps assms*(*3*) **by** *presburger*
from *Cons*(*2*) **have** *as*:∀ *a* ∈ *set As. a* ∈ *wf-pterm R*
unfolding *A* **by** *auto*
show *?case* **proof**(*cases f*)
case (*Inl* α)
from *Cons*(*2*) **have** *alpha*:*to-rule* α ∈ *R*
unfolding *A Inl* **using** *wf-pterm.simps* **by** *fastforce*
moreover from *Cons*(*2*) **have** *length As* = *length* (*var-rule* α)
unfolding *A Inl* **using** *wf-pterm.simps* **by** *fastforce*
ultimately show *?thesis*
unfolding *Inl A ctxt-of-pos-term.simps intp-actxt.simps* **using** *wf-pterm.intros*(*3*)[*OF alpha*] *IH as A*(*2*)
by (*smt* (*verit*, *ccfv-SIG*) *id-take-nth-drop in-set-conv-nth le-simps*(*1*) *length-append list.size*(*4*) *nth-append-take nth-append-take-drop-is-nth-conv*)
next
case (*Inr b*)
show *?thesis* **unfolding** *Inr A ctxt-of-pos-term.simps intp-actxt.simps* **using** *wf-pterm.intros*(*2*) *IH as A*(*2*)

by (*smt* (*verit*, *ccfv-SIG*) *Cons-nth-drop-Suc append-take-drop-id in-set-conv-nth length-append length-nth-simps*(*2*) *less-imp-le-nat nth-append-take nth-append-take-drop-is-nth-conv*)
qed
qed *simp*

2.4 'Normal' Terms vs. Proof Terms

lemma *to-pterm-empty*: *is-empty-step* (*to-pterm t*)
proof (*induction t*)
case (*Fun f ts*)
then have *list-all is-empty-step* (*map to-pterm ts*) **using** *list-all-iff* **by** *force*
then show *?case* **by** *simp*
qed *simp*

Variables remain unchanged.

lemma *vars-to-pterm*: *vars-term-list* (*to-pterm t*) = *vars-term-list t*
proof(*induction t*)
case (*Fun f ts*)
then have ∗:*map vars-term-list ts* = *map* (*vars-term-list* ∘ *to-pterm*) *ts* **by** *simp*
show *?case* **by** (*simp add*: ∗ *vars-term-list.simps*(*2*))
qed (*simp add*: *vars-term-list.simps*(*1*))

lemma *poss-list-to-pterm*: *poss-list t* = *poss-list* (*to-pterm t*)
proof(*induction t*)
case (*Fun f ts*)
then have ∗:*map poss-list ts* = *map* (*poss-list* ∘ *to-pterm*) *ts* **by** *simp*
show *?case* **by** (*simp add*: ∗ *poss-list.simps*(*2*))
qed (*simp add*: *poss-list.simps*(*1*))

lemma *p-in-poss-to-pterm*:
assumes *p* ∈ *poss t*
shows *p* ∈ *poss* (*to-pterm t*)
using *assms poss-list-to-pterm* **by** (*metis poss-list-sound*)

lemma *var-poss-to-pterm*: *var-poss t* = *var-poss* (*to-pterm t*)
proof(*induction t*)
case (*Fun f ts*)
then have ∗:*map var-poss ts* = *map* (*var-poss* ∘ *to-pterm*) *ts* **by** *simp*
then show *?case* **unfolding** *var-poss.simps to-pterm.simps*
by *auto*
qed *simp*

lemma *var-poss-list-to-pterm*: *var-poss-list* (*to-pterm t*) = *var-poss-list t*
proof(*induct t*)
case (*Fun f ts*)
then show *?case* **unfolding** *var-poss-list.simps to-pterm.simps*
by (*metis* (*no-types*, *lifting*) *length-map map-nth-eq-conv nth-mem*)
qed *simp*

to-pterm distributes over application of substitution.

lemma *to-pterm-subst*:
to-pterm $(t \cdot \sigma) = (\textit{to-pterm}\ t) \cdot (\textit{to-pterm} \circ \sigma)$
by (*induct t*, *auto*)

to-pterm distributes over context.

lemma *to-pterm-ctxt-of-pos-apply-term*:
assumes $p \in \textit{poss}\ s$
shows *to-pterm* $((\textit{ctxt-of-pos-term}\ p\ s)\ \langle t \rangle) = (\textit{ctxt-of-pos-term}\ p\ (\textit{to-pterm}\ s))\langle \textit{to-pterm}\ t \rangle$
using *assms* **proof**(*induct p arbitrary*:*s*)
case (*Cons i p*)
then obtain *f ss* **where** *s*:$s = \textit{Fun}\ f\ ss$ **and** *i*:$i < \textit{length}\ ss$ **and** *p*:$p \in \textit{poss}\ (ss!i)$
using *args-poss* **by** *blast*
then show *?case* **unfolding** *s to-pterm.simps ctxt-of-pos-term.simps intp-actxt.simps*
using *Cons*(*1*)
by (*simp add*: *drop-map take-map*)
qed *simp*

Linear terms become linear proof terms.

lemma *to-pterm-linear*:
assumes *linear-term t*
shows *linear-term* (*to-pterm t*)
using *assms* **proof**(*induction t*)
case (*Fun f ts*)
have ∗:*map vars-term ts* = *map vars-term* (*map to-pterm ts*)
by (*metis* (*mono-tags*, *lifting*) *length-map map-nth-eq-conv set-vars-term-list vars-to-pterm*)
with *Fun* **show** *?case* **by** *auto*
qed *simp*

lemma *lhs-subst-trivial*:
shows *match* (*to-pterm* (*lhs* α) $\cdot \langle As \rangle_\alpha$) (*to-pterm* (*lhs* α)) = *Some* $\langle As \rangle_\alpha$
using *match-trivial*
by (*smt comp-def mem-Collect-eq mk-subst-not-mem set-remdups set-rev set-vars-term-list subsetI subst-domain-def vars-to-pterm*)

lemma *to-pterm-ctxt-apply-term*:
to-pterm $C\langle t \rangle$ = (*to-pterm-ctxt C*) $\langle$*to-pterm t*$\rangle$
by(*induct C*) *simp-all*

2.5 Substitutions

lemma *fun-mk-subst*[*simp*]:
assumes $\forall x.\ f\ (\textit{Var}\ x) = \textit{Var}\ x$
shows $f \circ (\textit{mk-subst Var}\ (\textit{zip vs ts})) = \textit{mk-subst Var}\ (\textit{zip vs}\ (\textit{map f ts}))$
proof−
have $\forall a.\ f\ (\textit{case map-of}\ (\textit{zip vs ts})\ a\ \textit{of None} \Rightarrow \textit{Var}\ a \mid \textit{Some}\ t \Rightarrow t)$
$= (\textit{case map-of}\ (\textit{zip vs ts})\ a\ \textit{of None} \Rightarrow \textit{Var}\ a \mid \textit{Some}\ t \Rightarrow f\ t)$
using *assms* **by** (*simp add*: *option.case-eq-if*)

moreover have $\forall\, a.$ (*case map-of* (*zip vs* (*map f ts*)) *a of None* $\Rightarrow$ *Var a* | *Some* $x \Rightarrow x$)
$= (case\ (map\text{-}of\ (zip\ vs\ ts))\ a\ of\ None \Rightarrow Var\ a \mid Some\ t \Rightarrow f\ t)$
by (*simp add*:*zip-map2 map-of-map option.case-eq-if option.map-sel*)
ultimately show *?thesis* **unfolding** *mk-subst-def* **unfolding** *comp-def* **by** *auto*
qed

lemma *apply-lhs-subst-var-rule*:
assumes *length ts* = *length* (*var-rule* α)
shows *map* ($\langle ts \rangle_\alpha$) (*var-rule* α) = *ts*
using *assms* **by** (*simp add*: *mk-subst-distinct map-nth-eq-conv*)

lemma *match-lhs-subst*:
assumes *match B* (*to-pterm* (*lhs* α)) = *Some* σ
shows $\exists\, Bs.$ *length Bs* = *length* (*var-rule* α) $\wedge$
$B = (to\text{-}pterm\ (lhs\ \alpha)) \cdot \langle Bs \rangle_\alpha \wedge$
$(\forall\, x \in set\ (var\text{-}rule\ \alpha).\ \sigma\ x = (\langle Bs \rangle_\alpha)\ x)$
proof−
obtain *Bs* **where** *Bs*:*length Bs* = *length* (*var-rule* α)
$\forall\, i < length\ (var\text{-}rule\ \alpha).\ Bs!i = \sigma\ ((var\text{-}rule\ \alpha)!i)$
using *length-map nth-map* **by** *blast*
then have *2*:$(\forall\, x \in set\ (var\text{-}rule\ \alpha).\ \sigma\ x = (\langle Bs \rangle_\alpha)\ x)$
by (*smt apply-lhs-subst-var-rule in-set-idx nth-map*)
have *v*:*vars-term* (*to-pterm* (*lhs* α)) = *set* (*var-rule* α)
by (*metis comp-apply set-remdups set-rev set-vars-term-list vars-to-pterm*)
from *assms* **have** $B = (to\text{-}pterm\ (lhs\ \alpha)) \cdot \sigma$
using *match-matches* **by** *blast*
also have $\ldots = (to\text{-}pterm\ (lhs\ \alpha)) \cdot \langle Bs \rangle_\alpha$
by (*intro term-subst-eq*, *insert 2 v*, *auto*)
finally show *?thesis* **using** *Bs 2* **by** *auto*
qed

lemma *apply-subst-wf-pterm*:
assumes $A \in$ *wf-pterm R*
and $\forall\, x \in$ *vars-term A*. $\sigma\ x \in$ *wf-pterm R*
shows $A \cdot \sigma \in$ *wf-pterm R*
using *assms* **proof**(*induct A*)
case (*2 ts f*)
{fix *t* **assume** *t*:$t \in$ *set ts*
with *2*(*2*) **have** ($\forall\, x \in$*vars-term t*. $\sigma\ x \in$ *wf-pterm R*)
by (*meson term.set-intros*(*4*))
with *t 2*(*1*) **have** $t \cdot \sigma \in$ *wf-pterm R*
by *blast*
}
then show *?case* **unfolding** *eval-term.simps* **by** (*simp add*: *wf-pterm.intros*(*2*))
next
case (*3* α *As*)
{fix *a* **assume** *a*:$a \in$ *set As*
with *3*(*4*) **have** ($\forall\, x \in$*vars-term a*. $\sigma\ x \in$ *wf-pterm R*)

```
    by (meson term.set-intros(4))
   with a 3(3) have a · σ ∈ wf-pterm R
    by blast
 }
 with 3(1,2) show ?case unfolding eval-term.simps by (simp add: wf-pterm.intros(3))
qed simp

lemma subst-well-def:
 assumes B ∈ wf-pterm R A · σ = B x ∈ vars-term A
 shows σ x ∈ wf-pterm R
 using assms by (metis (no-types, lifting) poss-imp-subst-poss eval-term.simps(1)
subt-at-is-wf-pterm subt-at-subst vars-term-poss-subt-at)

lemma lhs-subst-args-wf-pterm:
 assumes to-pterm (lhs α) · ⟨As⟩α ∈ wf-pterm R and length As = length (var-rule
α)
 shows ∀ a ∈ set As. a ∈ wf-pterm R
proof−
 from assms have map (⟨As⟩α) (var-rule α) = As
   using apply-lhs-subst-var-rule by blast
 with assms show ?thesis
   by (smt comp-apply in-set-idx map-nth-eq-conv nth-mem set-remdups set-rev
set-vars-term-list subst-well-def vars-to-pterm)
qed

lemma match-well-def:
 assumes B ∈ wf-pterm R match B A = Some σ
 shows ∀ i < length (vars-distinct A). σ ((vars-distinct A) ! i) ∈ wf-pterm R
 using assms subst-well-def match-matches
 by (smt comp-apply nth-mem set-remdups set-rev set-vars-term-list)

lemma subst-imp-well-def:
 assumes A · σ ∈ wf-pterm R
 shows A ∈ wf-pterm R
 using assms proof(induct A)
 case (Pfun f As)
 {fix i assume i:i < length As
   with Pfun(2) have (As!i) · σ ∈ wf-pterm R
    by auto
   with Pfun(1) i have As!i ∈ wf-pterm R
    by simp
 }
 then show ?case using wf-pterm.intros(2)
   by (metis in-set-idx)
next
 case (Prule α As)
 {fix i assume i:i < length As
   with Prule(2) have (As!i) · σ ∈ wf-pterm R
    by auto
```

```
    with Prule(1) i have As!i ∈ wf-pterm R
      by simp
  }
  moreover from Prule(2) have to-rule α ∈ R length As = length (var-rule α)
    using wf-pterm.cases by force+
  ultimately show ?case using wf-pterm.intros(3) Prule(2)
    by (metis in-set-idx)
qed simp

lemma lhs-subst-var-i:
  assumes x = (var-rule α)!i and i < length (var-rule α) and i < length As
  shows (⟨As⟩α) x = As!i
 using assms mk-subst-distinct distinct-remdups by (metis comp-apply distinct-rev)

lemma lhs-subst-not-var-i:
  assumes ¬(∃ i < length As. i < length (var-rule α) ∧ x = (var-rule α)!i)
  shows (⟨As⟩α) x = Var x
  using assms proof(rule contrapos-np)
  {assume (⟨As⟩α) x ≠ Var x
    then obtain i where i < length (zip (var-rule α) As) and (var-rule α)!i = x
      unfolding mk-subst-def by (smt assms imageE in-set-zip map-of-eq-None-iff
option.case-eq-if)
    then show ∃ i<length As. i < length (var-rule α) ∧ x = var-rule α ! i
      by auto
  }
qed

lemma lhs-subst-upd:
  assumes length ss1 < length (var-rule α)
  shows ((⟨ss1 @ t # ss2⟩α) ((var-rule α)!(length ss1) := s)) = ⟨ss1 @ s # ss2⟩α
proof
  fix x
  show ((⟨ss1 @ t # ss2⟩α)(var-rule α ! length ss1 := s)) x = (⟨ss1 @ s # ss2⟩α)
x proof(cases x = (var-rule α)!(length ss1))
    case True
    with assms have ((⟨ss1 @ t # ss2⟩α)(var-rule α ! length ss1 := s)) x = s
      by simp
    moreover from assms have (⟨ss1 @ s # ss2⟩α) x = s unfolding True
       by (smt (verit, del-insts) add.commute add-Suc-right le-add-same-cancel2
le-imp-less-Suc length-append length-nth-simps(2) lhs-subst-var-i nth-append-length
zero-order(1))
    ultimately show ?thesis by simp
  next
    case False
    then show ?thesis
    by (smt (verit, del-insts) append-Cons-nth-not-middle fun-upd-apply length-append
length-nth-simps(2) lhs-subst-not-var-i lhs-subst-var-i)
  qed
qed
```

lemma *eval-lhs-subst*:
assumes *l*:*length* (*var-rule* α) = *length As*
shows (*to-pterm* (*lhs* α)) $\cdot$ $\langle As \rangle_\alpha \cdot \sigma$ = (*to-pterm* (*lhs* α)) $\cdot$ $\langle$*map* ($\lambda a.\ a \cdot \sigma$) *As*$\rangle_\alpha$
proof−
{fix *x* **assume** *x* ∈ *vars-term* (*to-pterm* (*lhs* α))
then obtain *i* **where** *i*:*i* < *length* (*var-rule* α) (*var-rule* α) !*i* = *x*
using *vars-to-pterm* **by** (*metis in-set-conv-nth set-vars-term-list vars-term-list-vars-distinct*)

with *l* **have** ($\langle As \rangle_\alpha$) *x* = *As*!*i*
by (*metis lhs-subst-var-i*)
then have *1*:($\langle As \rangle_\alpha \circ_s \sigma$) *x* = *As*!*i* $\cdot$ σ
unfolding *subst-compose-def* **by** *simp*
from *i l* **have** ($\langle$*map* ($\lambda a.\ a \cdot \sigma$) *As*$\rangle_\alpha$) *x* = *map* ($\lambda a.\ a \cdot \sigma$) *As* ! *i*
using *lhs-subst-var-i* **by** (*metis length-map*)
with *1 i l* **have** ($\langle As \rangle_\alpha \circ_s \sigma$) *x* = ($\langle$*map* ($\lambda a.\ a \cdot \sigma$) *As*$\rangle_\alpha$) *x* **by** *simp*
}
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *eval-same-vars-cong subst-subst-compose*)
qed

lemma *var-rule-pos-subst*:
assumes *i* < *length* (*var-rule* α) *length ss* = *length* (*var-rule* α)
and *p* ∈ *poss* (*lhs* α) *Var* ((*var-rule* α)!*i*) = (*lhs* α)|-*p*
shows *lhs* α $\cdot$ $\langle ss \rangle_\alpha$ |- (*p*@*q*) = (*ss*!*i*)|-*q*
proof−
from *assms*(*1*,*2*) **have** ($\langle ss \rangle_\alpha$) ((*var-rule* α)!*i*) = *ss*!*i*
using *lhs-subst-var-i* **by** *force*
with *assms*(*3*,*4*) **show** *?thesis* **by** *auto*
qed

lemma *lhs-subst-var-rule*:
assumes *vars-term t* ⊆ *vars-term* (*lhs* α)
shows *t* $\cdot$ $\langle$*map* σ (*var-rule* α)$\rangle_\alpha$ = *t* $\cdot$ σ
using *assms* **by** (*smt* (*verit*, *ccfv-SIG*) *apply-lhs-subst-var-rule comp-apply length-map map-eq-conv set-remdups set-rev set-vars-term-list subsetD term-subst-eq-conv*)

2.6 Contexts

lemma *match-lhs-context*:
assumes *i* < *length* (*vars-term-list t*) ∧ *p* = (*var-poss-list t*)!*i*
and *linear-term t*
and *match* (((*ctxt-of-pos-term p* (*t* $\cdot$ σ)))$\langle B \rangle$) *t* = *Some* τ
shows *map* τ (*vars-term-list t*) = (*map* σ (*vars-term-list t*))[*i* := *B*]
proof−
from *assms* **have** (*ctxt-of-pos-term p* (*t* $\cdot$ σ))$\langle B \rangle$ = *t* $\cdot$ (σ(*vars-term-list t*!*i* := *B*))
using *ctxt-apply-term-subst* **by** *blast*

with *assms*(3) **have** ∗:(∀ *x*∈*vars-term t*. (σ(*vars-term-list t*!*i* := *B*)) *x* = τ *x*)
using *match-complete′* **by** (*metis option.inject*)
from *assms*(2) **have** *distinct* (*vars-term-list t*)
by (*metis distinct-remdups distinct-rev linear-term-var-vars-term-list o-apply*)
with ∗ *assms*(1) **show** *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *fun-upd-other fun-upd-same length-list-update length-map map-nth-eq-conv nth-eq-iff-index-eq nth-list-update nth-mem set-vars-term-list*)
qed

lemma *ctxt-lhs-subst*:
assumes *i*:*i* < *length* (*var-poss-list* (*lhs* α)) **and** *l*:*length As* = *length* (*var-rule* α)
and *p1*:*p1* = *var-poss-list* (*lhs* α) ! *i* **and** *lin*:*linear-term* (*lhs* α)
and *p2* ∈ *poss* (*As*!*i*)
shows (*ctxt-of-pos-term* (*p1* @ *p2*) (*to-pterm* (*lhs* α) · ⟨*As*⟩α))⟨*A*⟩ =
(*to-pterm* (*lhs* α)) · ⟨*take i As* @ (*ctxt-of-pos-term p2* (*As*!*i*))⟨*A*⟩ # *drop* (*Suc i*) *As*⟩α
proof−
have *l2*:*length* (*var-poss-list* (*lhs* α)) = *length* (*var-rule* α)
using *lin* **by** (*metis length-var-poss-list linear-term-var-vars-term-list*)
from *p1 i* **have** *p1-pos*:*p1* ∈ *poss* (*to-pterm* (*lhs* α))
by (*metis nth-mem var-poss-imp-poss var-poss-list-sound var-poss-to-pterm*)
have *sub*:(*to-pterm* (*lhs* α))|-*p1* = *Var* (*vars-term-list* (*lhs* α)!*i*)
by (*metis i length-var-poss-list p1 var-poss-list-to-pterm vars-term-list-var-poss-list vars-to-pterm*)
have ∗∗: (*to-pterm* (*lhs* α) · ⟨*As*⟩α)|-*p1* = *As*!*i*
unfolding *subt-at-subst*[*OF p1-pos*] *sub eval-term.simps* **using** *i l l2* **by** (*metis lhs-subst-var-i lin linear-term-var-vars-term-list*)
then have ∗:(*ctxt-of-pos-term* (*p1* @ *p2*) (*to-pterm* (*lhs* α) · ⟨*As*⟩α)) = ((*ctxt-of-pos-term p1* (*to-pterm* (*lhs* α))) ·c ⟨*As*⟩α) ∘c (*ctxt-of-pos-term p2* (*As*!*i*))
using *ctxt-of-pos-term-append ctxt-of-pos-term-subst* **by** (*metis p1-pos poss-imp-subst-poss*)

show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) ∗ ∗∗ *ctxt-ctxt-compose*
ctxt-subst-apply lhs-subst-upd append-Cons-nth-not-middle
i id-take-nth-drop l l2 less-imp-le-nat lin linear-term-var-vars-term-list
nth-append-take p1-pos sub to-pterm-linear
ctxt-of-pos-term-append ctxt-supt-id eval-term.simps(1) *poss-imp-subst-poss*
replace-at-append-subst subt-at-subst)
qed

lemma *ctxt-rule-obtain-pos*:
assumes *iq*:*i*#*q* ∈ *poss* (*Prule* α *As*)
and *p-pos*:*p* ∈ *poss* (*source* (*Prule* α *As*))
and *ctxt*:*source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Prule* α *As*)) = *ctxt-of-pos-term p* (*source* (*Prule* α *As*))
and *lin*:*linear-term* (*lhs* α)
and *l*:*length As* = *length* (*var-rule* α)
shows ∃ *p1 p2*. *p* = *p1*@*p2* ∧ *p1* = *var-poss-list* (*lhs* α)!*i* ∧ *p2* ∈ *poss* (*source*

(*As*!*i*))
proof−
from *iq* **have** *i*:*i* < *length As*
by *simp*
let *?p1*=*var-poss-list* (*lhs* α)!*i*
have *p1*:(*var-poss-list* (*lhs* α) ! *length* (*take i As*)) = *?p1*
using *i* **by** *fastforce*
have *p1-pos*:*?p1* ∈ *poss* (*lhs* α)
by (*metis i l length-var-poss-list lin linear-term-var-vars-term-list nth-mem var-poss-imp-poss var-poss-list-sound*)
then have ∗:*source-ctxt* (*ctxt-of-pos-term* (*i* # *q*) (*Prule* α *As*)) = ((*ctxt-of-pos-term ?p1* (*lhs* α)) ·$_c$ ⟨*map source* (*take i As* @ *Var* (*vars-term-list* (*lhs* α) ! *length* (*take i As*)) # *drop* (*Suc i*) *As*)⟩$_\alpha$) ∘$_c$
source-ctxt (*ctxt-of-pos-term q* (*As* ! *i*))
unfolding *ctxt-of-pos-term.simps source-ctxt.simps Let-def p1* **by** (*simp add*: *ctxt-of-pos-term-subst*)
from *ctxt* **have** *?p1* ≤$_p$ *p*
unfolding ∗ **using** *p1-pos p-pos* **unfolding** *source.simps* **using** *ctxt-subst-comp-pos* **by** *blast*
then obtain *p2* **where** *p*:*p* = *?p1*@*p2*
using *prefix-def* **by** *blast*
have (*lhs* α)|-*?p1* = *Var* (*vars-term-list* (*lhs* α) !*i*)
by (*metis i l lin linear-term-var-vars-term-list vars-term-list-var-poss-list*)
moreover have *Var* (*vars-term-list* (*lhs* α) !*i*) · ⟨*map source As*⟩$_\alpha$ = *source* (*As*!*i*)
unfolding *eval-term.simps* **using** *lhs-subst-var-i i l* **by** (*smt* (*verit*, *best*) *length-map lin linear-term-var-vars-term-list nth-map*)
ultimately have *p2* ∈ *poss* (*source* (*As*!*i*))
using *p-pos* **unfolding** *p* **using** *p1-pos* **by** *auto*
with *p* **show** *?thesis* **by** *simp*
qed

2.7 Source and Target

lemma *source-empty-step*:
assumes *is-empty-step t*
shows *to-pterm* (*source t*) = *t*
using *assms* **by** (*induction t*) (*simp-all add*: *list-all-length map-nth-eq-conv*)

lemma *empty-coinitial*:
shows *co-initial A t* ⟹ *is-empty-step t* ⟹ *to-pterm* (*source A*) = *t*
by (*simp add*: *source-empty-step*)

lemma *source-to-pterm*[*simp*]: *source* (*to-pterm t*) = *t*
by (*induction t*) (*simp-all add*: *map-nth-eq-conv*)

lemma *target-to-pterm*[*simp*]: *target* (*to-pterm t*) = *t*
by (*induction t*) (*simp-all add*: *map-nth-eq-conv*)

lemma *vars-term-source*:
assumes $A \in$ *wf-pterm R*
shows *vars-term A* = *vars-term* (*source A*)
using *assms* **proof**(*induct A*)
case (*3 α As*)
show *?case* **proof**
{**fix** *x* **assume** $x \in$ *vars-term* (*Prule α As*)
then obtain *i* **where** *i*:$i <$ *length As x* $\in$ *vars-term* (*As*!*i*)
by (*metis term.sel*(*4*) *var-imp-var-of-arg*)
from *i*(*1*) *3*(*2*) **obtain** *j* **where** *j*:$j <$ *length* (*vars-term-list* (*lhs α*)) *vars-term-list* (*lhs α*)!*j* = *var-rule α* !*i*
by (*metis comp-apply in-set-idx nth-mem set-remdups set-rev*)
let *?p*=(*var-poss-list* (*lhs α*)!*j*)
from *j* **have** *p*:*?p* $\in$ *poss* (*lhs α*)
by (*metis in-set-conv-nth length-var-poss-list var-poss-imp-poss var-poss-list-sound*)

with *3*(*2*) *i*(*1*) *j* **have** *source* (*Prule α As*) |- *?p* = *source* (*As*!*i*)
using *mk-subst-distinct* **unfolding** *source.simps*
by (*smt* (*verit*, *best*) *comp-apply distinct-remdups distinct-rev filter-cong length-map map-nth-conv mk-subst-same eval-term.simps*(*1*) *subt-at-subst vars-term-list-var-poss-list*)

with *3*(*3*) **have** *x* $\in$ *vars-term* (*source* (*Prule α As*))
unfolding *source.simps* **using** *vars-term-subt-at p*
by (*smt* (*verit*, *ccfv-SIG*) *i nth-mem poss-imp-subst-poss subsetD*)
}
then show *vars-term* (*Prule α As*) $\subseteq$ *vars-term* (*source* (*Prule α As*))
by *blast*
{**fix** *x* **assume** $x \in$ *vars-term* (*source* (*Prule α As*))
then obtain *y* **where** *y*:$y \in$ *vars-term* (*lhs α*) $x \in$ *vars-term* ((⟨*map source As*⟩$_\alpha$) *y*)
using *vars-term-subst* **by** *force*
then obtain *i* **where** *i*:$i <$ *length* (*var-rule α*) *y* = *var-rule α*!*i*
by (*metis in-set-idx set-vars-term-list vars-term-list-vars-distinct*)
with *y*(*2*) *3*(*2*) **have** *x* $\in$ *vars-term* (*source* (*As*!*i*))
by (*simp add*: *mk-subst-distinct*)
with *3 i*(*1*) **have** *x* $\in$ *vars-term* (*Prule α As*)
by (*metis nth-mem term.set-intros*(*4*))
}
then show *vars-term* (*source* (*Prule α As*)) $\subseteq$ *vars-term* (*Prule α As*)
by *blast*
qed
qed *auto*

context *var-rhs-subset-lhs*
begin
lemma *vars-term-target*:
assumes $A \in$ *wf-pterm R*
shows *vars-term* (*target A*) $\subseteq$ *vars-term A*
using *assms* **proof**(*induct A*)

```
  case (3 α As)
  show ?case proof
  fix x assume x ∈ vars-term (target (Prule α As))
    then obtain y where y:y ∈ vars-term (rhs α)  x ∈ vars-term ((⟨map target
As⟩α) y)
      using vars-term-subst by force
    then have y ∈ vars-term (lhs α)
      using 3.hyps(1) varcond by auto
    then obtain i where i:i < length (var-rule α) y = var-rule α!i
      by (metis in-set-idx set-vars-term-list vars-term-list-vars-distinct)
    with y(2) 3(2) have x ∈ vars-term (target (As!i))
      by (simp add: mk-subst-distinct)
    with 3 i(1) show x ∈ vars-term (Prule α As)
      by fastforce
  qed
qed auto
end

lemma linear-source-imp-linear-pterm:
  assumes A ∈ wf-pterm R linear-term (source A)
  shows linear-term A
  using assms proof(induct A)
  case (2 As f)
 then show ?case unfolding source.simps linear-term.simps using vars-term-source
   by (smt (verit, ccfv-SIG) in-set-idx length-map map-equality-iff nth-map nth-mem)
next
  case (3 α As)
  {fix a assume a:a ∈ set As
    with 3(2) obtain i where i:i < length (var-rule α) As!i = a
      by (metis in-set-idx)
    let ?x=var-rule α ! i
    from i have ?x ∈ vars-term (lhs α)
      by (metis comp-apply nth-mem set-remdups set-rev set-vars-term-list)
    then obtain p where p ∈ poss (lhs α) lhs α |-p = Var ?x
      by (meson vars-term-poss-subt-at)
    then have source (Prule α As) ⊵ source a
      unfolding source.simps using lhs-subst-var-i[of ?x α i As] i 3(2)
     by (smt (verit, best) ‹var-rule α ! i ∈ vars-term (lhs α)› apply-lhs-subst-var-rule
eval-term.simps(1) length-map map-nth-conv supteq-subst vars-term-supteq)
    then have linear-term (source a)
      using 3(4) by (metis subt-at-linear supteq-imp-subt-at)
    with 3(3) a have linear-term a by simp
  }
  moreover have is-partition (map vars-term As) proof−
    {fix i j assume i:i < length As and j:j < length As and ij:i ≠ j
      let ?x=var-rule α ! i and ?y=var-rule α ! j
      from i j ij 3(2) have xy:?x ≠ ?y
        by (simp add: nth-eq-iff-index-eq)
      from i 3(2) have ?x ∈ vars-term (lhs α)
```

by (*metis comp-apply nth-mem set-remdups set-rev set-vars-term-list*)
then obtain *p* **where** *p*:*p* ∈ *poss* (*lhs* α) *lhs* α |-*p* = *Var ?x*
by (*meson vars-term-poss-subt-at*)
from *j 3*(*2*) **have** *?y* ∈ *vars-term* (*lhs* α)
by (*metis comp-apply nth-mem set-remdups set-rev set-vars-term-list*)
then obtain *q* **where** *q*:*q* ∈ *poss* (*lhs* α) *lhs* α |-*q* = *Var ?y*
by (*meson vars-term-poss-subt-at*)
from *xy p q* **have** *p* ⊥ *q*
using *prefix-def parallel-pos*
by (*metis self-append-conv term.inject*(*1*) *var-pos-maximal*)
moreover have *source* (*Prule* α *As*) |-*p* = *source* (*As*!*i*)
unfolding *source.simps* **by** (*metis* (*mono-tags*, *lifting*) *3.hyps*(*2*) *eval-term.simps*(*1*) *i length-map lhs-subst-var-i nth-map p subt-at-subst*)
moreover have *source* (*Prule* α *As*) |-*q* = *source* (*As*!*j*)
unfolding *source.simps* **by** (*metis* (*mono-tags*, *lifting*) *3.hyps*(*2*) *eval-term.simps*(*1*) *j length-map lhs-subst-var-i nth-map q subt-at-subst*)
ultimately have *vars-term* (*source* (*As*!*i*)) ∩ *vars-term* (*source* (*As*!*j*)) = {}
using *3*(*4*) **by** (*metis linear-subterms-disjoint-vars p*(*1*) *poss-imp-subst-poss q*(*1*) *source.simps*(*3*))
then have *vars-term* (*As*!*i*) ∩ *vars-term* (*As*!*j*) = {}
using *vars-term-source 3*(*3*) *i j* **using** *nth-mem* **by** *blast*
}
then show *?thesis*
unfolding *is-partition-alt is-partition-alt-def* **by** *simp*
qed
ultimately show *?case* **unfolding** *source.simps linear-term.simps* **by** *simp*
qed *simp*

context *var-rhs-subset-lhs*
begin
lemma *target-apply-subst*:
assumes *A* ∈ *wf-pterm R*
shows *target* (*A* · σ) = (*target A*) · (*target* ∘ σ)
using *assms*(*1*) **proof**(*induct A*)
case (*2 ts f*)
then have (*map target* (*map* (λ*t*. *t* · σ) *ts*)) = (*map* (λ*t*. *t* · (*target* ∘ σ)) (*map target ts*))
unfolding *map-map o-def* **by** *auto*
then show *?case* **unfolding** *eval-term.simps target.simps* **by** *simp*
next
case (*3* α *As*)
have *id*:∀ *x* ∈ *vars-term* (*rhs* α). (⟨*map* (*target* ∘ (λ*t*. *t* · σ)) *As*⟩$_\alpha$) *x* = (⟨*map target As*⟩$_\alpha$ ∘$_s$ (*target* ∘ σ)) *x*
proof−
have *vars*:*vars-term* (*rhs* α) ⊆ *set* (*var-rule* α)
using *3*(*1*) *varcond* **by** *auto*
{ **fix** *i* **assume** *i*:*i* < *length* (*var-rule* α)
with *3* **have** (⟨*map* (*target* ∘ (λ*t*. *t* · σ)) *As*⟩$_\alpha$) ((*var-rule* α)!*i*) = *target* ((*As*!*i*) · σ)

```
    by (simp add: mk-subst-distinct)
  also have ... = target (As!i) · (target ∘ σ)
    using 3 i by (metis nth-mem)
  also have ... = (⟨map target As⟩α ∘s (target ∘ σ)) ((var-rule α)!i)
    using 3 i unfolding subst-compose-def by (simp add: mk-subst-distinct)
  finally have (⟨map (target ∘ (λt. t · σ)) As⟩α) ((var-rule α)!i) = (⟨map target
As⟩α ∘s (target ∘ σ)) ((var-rule α)!i) .
 } with vars show ?thesis by (smt (z3) in-mono in-set-conv-nth)
 qed
 have target ((Prule α As) · σ) = (rhs α) · ⟨map (target ∘ (λt. t · σ)) As⟩α
  unfolding eval-term.simps(2) by simp
 also have ... = (rhs α) · (⟨map target As⟩α ∘s (target ∘ σ))
  using id by (meson term-subst-eq)
 also have ... = (target (Prule α As)) · (target ∘ σ) by simp
 finally show ?case .
qed simp
end

context var-rhs-subset-lhs
begin
lemma tgt-subst-simp:
assumes A ∈ wf-pterm R
 shows target (A · σ) = target ((to-pterm (target A)) · σ)
 by (metis assms target-apply-subst target-to-pterm to-pterm-wf-pterm)
end

lemma target-empty-apply-subst:
 assumes is-empty-step t
 shows target (t · σ) = (target t) · (target ∘ σ)
using assms proof(induction t)
 case (Var x)
 then show ?case by (metis comp-apply eval-term.simps(1) target.simps(1))
next
 case (Pfun f As)
 from Pfun(2) have ∀ a ∈ set As. is-empty-step a
  by (simp add: Ball-set-list-all)
 with Pfun(1) show ?case by simp
next
 case (Prule α As)
 then show ?case
  using is-empty-step.simps(3) by blast
qed

lemma source-ctxt-comp:source-ctxt (C1 ∘c C2) = source-ctxt C1 ∘c source-ctxt
C2
 by(induct C1) (simp-all add:ctxt-monoid-mult.mult-assoc)

lemma context-source: source (A⟨B⟩) = source (A⟨to-pterm (source B)⟩)
proof(induct A rule:actxt.induct)
```

case (*More f ss1 A ss2*)
then show *?case* **by**(*cases f*) *simp-all*
qed *simp*

lemma *context-target*: *target* (*A*⟨*B*⟩) = *target* (*A*⟨*to-pterm* (*target B*)⟩)
proof(*induct A rule*:*actxt.induct*)
case (*More f ss1 A ss2*)
then show *?case* **by**(*cases f*) *simp-all*
qed *simp*

lemma *source-to-pterm-ctxt*:
source ((*to-pterm-ctxt C*)⟨*A*⟩) = *C*⟨*source A*⟩
by (*metis context-source source-to-pterm to-pterm-ctxt-apply-term*)

lemma *target-to-pterm-ctxt*:
target ((*to-pterm-ctxt C*)⟨*A*⟩) = *C*⟨*target A*⟩
by (*metis context-target target-to-pterm to-pterm-ctxt-apply-term*)

lemma *source-ctxt-to-pterm*:
assumes *p* ∈ *poss s*
shows *source-ctxt* (*ctxt-of-pos-term p* (*to-pterm s*)) = *ctxt-of-pos-term p s*
using *assms* **proof**(*induct p arbitrary*:*s*)
case (*Cons i p*)
then obtain *f ss* **where** *s*:*s* = *Fun f ss* **and** *i* < *length ss* **and** *p* ∈ *poss* (*ss*!*i*)
using *args-poss* **by** *blast*
then show *?case*
unfolding *s to-pterm.simps ctxt-of-pos-term.simps source-ctxt.simps* **using** *Cons*(*1*)
by (*smt* (*verit*, *best*) *drop-map nth-map source.simps*(*2*) *source-to-pterm take-map term.inject*(*2*) *to-pterm.simps*(*2*))
qed *simp*

lemma *to-pterm-ctxt-at-pos*:
assumes *p* ∈ *poss s*
shows *ctxt-of-pos-term p* (*to-pterm s*) = *to-pterm-ctxt* (*ctxt-of-pos-term p s*)
using *assms* **proof**(*induct p arbitrary*:*s*)
case (*Cons i p*)
then obtain *f ss* **where** *s*:*s* = *Fun f ss*
using *args-poss* **by** *blast*
with *Cons* **show** *?case*
using *drop-map s take-map* **by** *force*
qed *simp*

lemma *to-pterm-ctxt-hole-pos*: *hole-pos C* = *hole-pos* (*to-pterm-ctxt C*)
by(*induct C*) *simp-all*

lemma *source-to-pterm-ctxt′*:
assumes *q* ∈ *poss s*
shows *source-ctxt* (*to-pterm-ctxt* (*ctxt-of-pos-term q s*)) = *ctxt-of-pos-term q s*

using *assms* **proof**(*induct q arbitrary*: *s*)
case (*Cons i q*)
then obtain *f ss* **where** *s*:*s* = *Fun f ss* **and** *i*:*i* < *length ss*
by (*meson args-poss*)
with *Cons* **have** *IH*:*source-ctxt* (*to-pterm-ctxt* (*ctxt-of-pos-term q* (*ss*!*i*))) = *ctxt-of-pos-term q* (*ss*!*i*)
by *auto*
with *i* **show** *?case* **unfolding** *s ctxt-of-pos-term.simps to-pterm-ctxt.simps source-ctxt.simps*
using *source-to-pterm* **by** (*metis source.simps*(*2*) *term.sel*(*4*) *to-pterm.simps*(*2*))

qed *simp*

lemma *to-pterm-ctxt-comp*: *to-pterm-ctxt* (*C* ∘$_c$ *D*) = *to-pterm-ctxt C* ∘$_c$ *to-pterm-ctxt D*
by(*induct C*) *simp-all*

lemma *source-apply-subst*:
assumes *A* ∈ *wf-pterm R*
shows *source* (*A* · *σ*) = (*source A*) · (*source* ∘ *σ*)
using *assms* **proof**(*induct A*)
case (*3 α As*)
have *id*:∀ *x* ∈ *vars-term* (*lhs α*). (⟨*map* (*source* ∘ (λ*t*. *t* · *σ*)) *As*⟩$_\alpha$) *x* = (⟨*map source As*⟩$_\alpha$ ∘$_s$ (*source* ∘ *σ*)) *x*
proof−
have *vars*:*vars-term* (*lhs α*) = *set* (*var-rule α*) **by** *simp*
{ fix *i* **assume** *i*:*i* < *length* (*var-rule α*)
with *3* **have** (⟨*map* (*source* ∘ (λ*t*. *t* · *σ*)) *As*⟩$_\alpha$) ((*var-rule α*)!*i*) = *source* ((*As*!*i*) · *σ*)
by (*simp add*: *mk-subst-distinct*)
also have ... = *source* (*As*!*i*) · (*source* ∘ *σ*)
using *3 i* **by** (*metis nth-mem*)
also have ... = (⟨*map source As*⟩$_\alpha$ ∘$_s$ (*source* ∘ *σ*)) ((*var-rule α*)!*i*)
using *3 i* **unfolding** *subst-compose-def* **by** (*simp add*: *mk-subst-distinct*)
finally have (⟨*map* (*source* ∘ (λ*t*. *t* · *σ*)) *As*⟩$_\alpha$) ((*var-rule α*)!*i*) = (⟨*map source As*⟩$_\alpha$ ∘$_s$ (*source* ∘ *σ*)) ((*var-rule α*)!*i*) **.**
} with *vars* **show** *?thesis* **by** (*metis in-set-idx*)
qed
have *source* ((*Prule α As*) · *σ*) = (*lhs α*) · ⟨*map* (*source* ∘ (λ*t*. *t* · *σ*)) *As*⟩$_\alpha$
unfolding *eval-term.simps*(*2*) **by** *simp*
also have ... = (*lhs α*) · (⟨*map source As*⟩$_\alpha$ ∘$_s$ (*source* ∘ *σ*))
using *id* **by** (*meson term-subst-eq*)
also have ... = (*source* (*Prule α As*)) · (*source* ∘ *σ*) **by** *simp*
finally show *?case* **.**
qed *simp-all*

lemma *ctxt-of-pos-term-at-var-subst*:
assumes *linear-term t*
and *p* ∈ *poss t* **and** *t*|-*p* = *Var x*
and ∀ *y* ∈ *vars-term t*. *y* ≠ *x* ⟶ *τ y* = *σ y*

shows *ctxt-of-pos-term p* $(t \cdot \tau)$ = *ctxt-of-pos-term p* $(t \cdot \sigma)$
using *assms* **proof**(*induct t arbitrary*:*p*)
case (*Fun f ts*)
from *Fun*(*3*,*4*) **obtain** *i p'* **where** *p*:*p* = *i*#*p'* **and** *i*:*i* < *length ts* **and** *p'*:*p'* ∈ *poss* (*ts*!*i*)
by *auto*
with *Fun*(*4*) **have** *x*:*ts*!*i* |*-p'* = *Var x*
by *simp*
{fix *j* **assume** *j*:*j* < *length ts j* ≠ *i*
from *Fun*(*2*) **have** *x* ∉ *vars-term* (*ts*!*j*)
by (*metis i j p' subset-eq term.set-intros*(*3*) *var-in-linear-args vars-term-subt-at x*)
with *Fun*(*5*) *j* **have** *ts*!*j* · *τ* = *ts*!*j* · *σ*
by (*metis* (*no-types*, *lifting*) *nth-mem term.set-intros*(*4*) *term-subst-eq*)
then have (*map* (*λt. t* · *τ*) *ts*)!*j* = (*map* (*λt. t* · *σ*) *ts*)!*j*
by (*simp add*: *j*)
}note *args*=*this*
from *args* **have** *args1*:*take i* (*map* (*λt. t* · *τ*) *ts*) = *take i* (*map* (*λt. t* · *σ*) *ts*)
using *nth-take-lemma*[*of i* (*map* (*λt. t* · *τ*) *ts*) (*map* (*λt. t* · *σ*) *ts*)] *i* **by** *simp*
from *args* **have** *args2*:*drop* (*Suc i*) (*map* (*λt. t* · *τ*) *ts*) = *drop* (*Suc i*) (*map* (*λt. t* · *σ*) *ts*)
using *nth-drop-equal*[*of* (*map* (*λt. t* · *τ*) *ts*) (*map* (*λt. t* · *σ*) *ts*) *Suc i*] *i* **by** *simp*
from *Fun*(*1*,*2*,*5*) *i* **have** *IH*:*ctxt-of-pos-term p'* ((*ts*!*i*) · *τ*) = *ctxt-of-pos-term p'* ((*ts*!*i*) · *σ*)
by (*simp add*: *p' x*)
with *args1 args2* **show** *?case*
unfolding *p eval-term.simps ctxt-of-pos-term.simps* **by** (*simp add*: *i*)
qed *simp*

context *left-lin*
begin

lemma *source-ctxt-apply-subst*:
assumes *C* ∈ *wf-pterm-ctxt R*
shows *source-ctxt* (*C* ·$_c$ *σ*) = (*source-ctxt C*) ·$_c$ (*source* ∘ *σ*)
using *assms* **proof**(*induct C*)
case (*2 ss1 ss2 C f*)
then show *?case*
unfolding *source-ctxt.simps actxt.simps 2* **using** *source-apply-subst* **by** *auto*
next
case (*3 α ss1 ss2 C*)
let *?p*=(*var-poss-list* (*lhs α*) ! *length ss1*)
let *?x*=(*vars-term-list* (*lhs α*) ! *length ss1*)
have *var-at-p*:(*lhs α*)|*-?p* = *Var ?x*
by (*metis 3.hyps*(*2*) *add-lessD1 length-remdups-leq length-rev less-add-one o-apply order-less-le-trans vars-term-list-var-poss-list*)
from *3*(*2*) **have** *pos1*:*?p* ∈ *poss* (*lhs α*)
by (*metis add-lessD1 comp-apply length-remdups-leq length-rev length-var-poss-list*

less-add-one nth-mem order-less-le-trans var-poss-imp-poss var-poss-list-sound)
then have *pos*:*?p* ∈ *poss* (*lhs* α · ⟨*map source* (*ss1* @ *Var* (*vars-term-list* (*lhs* α) ! *length ss1*) # *ss2*)⟩α)
using *poss-imp-subst-poss* **by** *blast*
have *lin*:*linear-term* (*lhs* α)
using *3*(*1*) *left-lin* **using** *left-linear-trs-def* **by** *fastforce*
{fix *y* **assume** *y* ∈ *vars-term* (*lhs* α) **and** *x*:*y* ≠ *?x*
then obtain *i* **where** *i*:*i* < *length* (*var-rule* α) *var-rule* α ! *i* = *y*
by (*metis in-set-idx lin linear-term-var-vars-term-list set-vars-term-list*)
with *x* **consider** *i* < *length ss1* | *i* > *length ss1* ∧ *i* < *length* (*var-rule* α)
using *lin linear-term-var-vars-term-list nat-neq-iff* **by** *fastforce*
then have (⟨*map source* (*map* (λ*t*. *t* · σ) *ss1* @ *Var ?x* # *map* (λ*t*. *t* · σ) *ss2*)⟩α) *y* = ((⟨*map source* (*ss1* @ *Var ?x* # *ss2*)⟩α) *y*) · (*source* ∘ σ)
proof(*cases*)
case *1*
with *i* **have** (⟨*map source* (*map* (λ*t*. *t* · σ) *ss1* @ *Var ?x* # *map* (λ*t*. *t* · σ) *ss2*)⟩α) *y* = *source* ((*ss1*!*i*) · σ)
by (*smt* (*z3*) *3.hyps*(*2*) *One-nat-def add.right-neutral add-Suc-right append-Cons-nth-left comp-apply distinct-remdups distinct-rev length-append length-map length-nth-simps*(*2*) *map-nth-eq-conv mk-subst-same*)
moreover from *i 1* **have** (⟨*map source* (*ss1* @ *Var ?x* # *ss2*)⟩α) *y* = *source* (*ss1*!*i*)
by (*smt* (*verit*, *ccfv-threshold*) *3.hyps*(*2*) *One-nat-def ab-semigroup-add-class.add-ac*(*1*) *append-Cons-nth-left comp-apply distinct-remdups distinct-rev length-append length-map list.size*(*4*) *map-nth-eq-conv mk-subst-distinct*)
moreover have *ss1*!*i* ∈ *wf-pterm R*
using *3*(*3*) *1* **by** (*meson UnCI nth-mem*)
ultimately show *?thesis*
using *source-apply-subst* **by** *auto*
next
case *2*
let *?i*=*i* − ((*length ss1*)+*1*)
have *i'*:*?i* < *length ss2*
using *3*(*2*) *2* **by** (*simp add*: *less-diff-conv2*)
have *i1*:(*map source* (*map* (λ*t*. *t* · σ) *ss1* @ *Var ?x* # *map* (λ*t*. *t* · σ) *ss2*))!*i* = *source* ((*ss2*!*?i*) · σ) **proof**−
have *i''*:*i* = *length* (*map source* (*map* (λ*t*. *t* · σ) *ss1* @ [*Var ?x*])) + *?i*
unfolding *length-append length-map* **using** *2* **by** *force*
show *?thesis* **unfolding** *map-append list.map*
using *i' i'' nth-append-length-plus*[*of* (*map source* (*map* (λ*t*. *t* · σ) *ss1* @ [*Var* (*vars-term-list* (*lhs* α) ! *length ss1*)])) *map source* (*map* (λ*t*. *t* · σ) *ss2*)]
by (*smt* (*verit*, *del-insts*) *Cons-eq-appendI append-Nil append-assoc length-map list.simps*(*9*) *map-append nth-map*)
qed
have *i2*:*map source* (*ss1* @ *Var ?x* # *ss2*) ! *i* = *source* (*ss2*!*?i*) **proof**−
have *i''*:*i* = *length* (*map source* (*ss1* @ [*Var ?x*])) + *?i*
unfolding *length-append length-map* **using** *2* **by** *force*
show *?thesis* **unfolding** *map-append list.map*
using *i' i'' nth-append-length-plus*[*of* (*map source* (*ss1* @ [*Var* (*vars-term-list*

(*lhs* α) ! *length ss1*)])) *map source ss2*]
by (*smt* (*verit*, *del-insts*) *append.left-neutral append-Cons append-assoc length-map list.simps*(*9*) *map-append nth-map*)
qed
from *i1 2* **have** (⟨*map source* (*map* (λ*t*. *t* · σ) *ss1* @ *Var ?x* # *map* (λ*t*. *t* · σ) *ss2*)⟩$_\alpha$) *y* = *source* ((*ss2*!*?i*) · σ)
by (*smt* (*verit*, *ccfv-threshold*) *3.hyps*(*2*) *One-nat-def ab-semigroup-add-class.add-ac*(*1*) *comp-def distinct-remdups distinct-rev i*(*2*) *length-append length-map list.size*(*4*) *mk-subst-distinct*)
moreover from *i2 2* **have** (⟨*map source* (*ss1* @ *Var ?x* # *ss2*)⟩$_\alpha$) *y* = *source* (*ss2*!*?i*)
by (*metis* (*no-types*, *opaque-lifting*) *3.hyps*(*2*) *One-nat-def add.right-neutral add-Suc-right comp-apply distinct-remdups distinct-rev i*(*2*) *length-append length-map length-nth-simps*(*2*) *mk-subst-distinct*)
moreover have *ss2*!*?i* ∈ *wf-pterm R*
using *3*(*3*) *2* ‹*?i* < *length ss2*› **by** (*metis UnCI nth-mem*)
ultimately show *?thesis*
using *source-apply-subst* **by** *auto*
qed
}
then have *ctxt-of-pos-term ?p* (*lhs* α · ⟨*map source* (*map* (λ*t*. *t* · σ) *ss1* @ *Var ?x* # *map* (λ*t*. *t* · σ) *ss2*)⟩$_\alpha$) =
ctxt-of-pos-term ?p (*lhs* α · ⟨*map source* (*ss1* @ *Var ?x* # *ss2*)⟩$_\alpha$ · (*source* ∘ σ))
using *ctxt-of-pos-term-at-var-subst*[*OF lin pos1 var-at-p*] **unfolding** *subst-subst*
by (*smt* (*verit*) *subst-compose*)
then show *?case* **unfolding** *source-ctxt.simps actxt.simps Let-def 3 subst-compose-ctxt-compose-distrib length-map ctxt-of-pos-term-subst*[*OF pos*, *symmetric*]
by *presburger*
qed *simp*

Needs left-linearity to avoid multihole contexts.

lemma *source-ctxt-apply-term*:
assumes *C* ∈ *wf-pterm-ctxt R*
shows *source* (*C*⟨*A*⟩) = (*source-ctxt C*)⟨*source A*⟩
using *assms* **proof**(*induct C*)
case (*3* α *ss1 ss2 C*)
from *3*(*1*) *left-lin* **have** *lin*:*linear-term* (*lhs* α)
using *left-linear-trs-def* **by** *fastforce*
from *3*(*2*) **have** *len*:*length ss1* < *length* (*vars-term-list* (*lhs* α))
by (*metis add-lessD1 less-add-one lin linear-term-var-vars-term-list*)
have (*source-ctxt* (*Crule* α *ss1 C ss2*))⟨*source A*⟩ =
lhs α · ⟨(*map source ss1*) @ (*source-ctxt C*)⟨*source A*⟩ # (*map source ss2*)⟩$_\alpha$
unfolding *source-ctxt.simps Let-def intp-actxt.simps source.simps ctxt-ctxt-compose*
using *ctxt-apply-term-subst*[*OF lin len*] *lhs-subst-upd*
by (*smt* (*verit*) *len length-map lin linear-term-var-vars-term-list list.simps*(*9*) *map-append*)
with *3*(*5*) **show** *?case* **by** *simp*
qed *simp-all*

end

lemma *rewrite-tgt*:
assumes *rstep*:$(t,v) \in (rstep\ R)^*$
shows $(target\ (C\ \langle(to\text{-}pterm\ t) \cdot \sigma\rangle), target\ (C\ \langle(to\text{-}pterm\ v) \cdot \sigma\rangle)) \in (rstep\ R)^*$
proof(*induct C*)
case *Hole*
then show *?case*
by (*simp add*: *local.rstep rsteps-closed-subst target-empty-apply-subst to-pterm-empty*)

next
case (*Cfun f ss1 C ss2*)
then show *?case* **by** (*simp add*: *ctxt-closed-one ctxt-closed-rsteps*)
next
case (*Crule α ss1 C ss2*)
let *?ts*=*map target* (*ss1* @ $C\langle to\text{-}pterm\ t \cdot \sigma\rangle$ # *ss2*)
let *?vs*=*map target* (*ss1* @ $C\langle to\text{-}pterm\ v \cdot \sigma\rangle$ # *ss2*)
{fix *x* **assume** $x \in vars\text{-}term\ (rhs\ \alpha)$
from *Crule* **have** $((\langle ?ts\rangle_\alpha)\ x, (\langle ?vs\rangle_\alpha)\ x) \in (rstep\ R)^*$
proof(*cases* $\exists\, i{<}length\ ?ts.\ i < length\ (var\text{-}rule\ \alpha) \wedge x = var\text{-}rule\ \alpha\ !\ i$)
case *True*
then obtain *i* **where** *i*:$i < length\ ?ts\ i < length\ ?vs\ i < length\ (var\text{-}rule\ \alpha)$
$x = var\text{-}rule\ \alpha\ !\ i$
by *auto*
show *?thesis* **using** *Crule* **unfolding** *lhs-subst-var-i*[*OF i*(*4*,*3*,*1*)] *lhs-subst-var-i*[*OF i*(*4*,*3*,*2*)]
nth-map[*OF i*(*1*)[*unfolded length-map*]] *nth-map*[*OF i*(*2*)[*unfolded length-map*]]
by (*metis append-Cons-nth-not-middle nth-append-length rtrancl.rtrancl-refl*)

next
case *False*
then have $*$:$\neg(\exists\, i{<}length\ ?vs.\ i < length\ (var\text{-}rule\ \alpha) \wedge x = var\text{-}rule\ \alpha\ !\ i)$
by *simp*
show *?thesis*
unfolding *lhs-subst-not-var-i*[*OF False*] *lhs-subst-not-var-i*[*OF* $*$] **by** *simp*
qed
}
then show *?case* **by** (*simp add*: *subst-rsteps-imp-rsteps*)
qed

2.8 Additional Results

lemma *length-args-well-Prule*:
assumes *Prule α As* ∈ *wf-pterm R Prule α Bs* ∈ *wf-pterm S*
shows *length As* = *length Bs*
proof−
from *assms*(*1*) **have** *length As* = *length* (*var-rule α*) **using** *wf-pterm.simps* **by** *fastforce*
moreover from *assms*(*2*) **have** *length Bs* = *length* (*var-rule α*) **using** *wf-pterm.simps*

by *fastforce*
ultimately show *?thesis* **by** *simp*
qed

lemma *co-initial-Var*:
assumes *co-initial* (*Var x*) *B*
shows $B = Var\ x \vee (\exists \alpha\ b'\ y.\ B = Prule\ \alpha\ b' \wedge lhs\ \alpha = Var\ y)$
proof−
{assume $B \neq Var\ x$
with *assms* **obtain** $\alpha\ b'$ **where** $B = Prule\ \alpha\ b'$
by (*metis is-empty-step.elims*(*3*) *source.elims source-empty-step term.distinct*(*1*))
moreover with *assms* **have** $\exists\ y.\ lhs\ \alpha = Var\ y$
by (*metis source.simps*(*1*) *source.simps*(*3*) *subst-apply-eq-Var*)
ultimately have $(\exists \alpha\ b'\ y.\ B = Prule\ \alpha\ b' \wedge lhs\ \alpha = Var\ y)$
by *blast*
}
then show *?thesis*
by *blast*
qed

lemma *source-poss*:
assumes *p*:$p \in poss\ (source\ (Pfun\ f\ As))$ **and** *iq*:$i\#q \in poss\ (Pfun\ f\ As)$
and *ctxt*:*source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Pfun f As*)) = *ctxt-of-pos-term p* (*source* (*Pfun f As*))
shows $\exists p'.\ p = i\#p' \wedge p' \in poss\ (source\ (As!i))$
proof−
obtain p' **where** *hole-pos* (*source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Pfun f As*))) = $i\#p'$
$p' = hole\text{-}pos\ (source\text{-}ctxt\ (ctxt\text{-}of\text{-}pos\text{-}term\ q\ (As\ !\ i)))$
unfolding *ctxt-of-pos-term.simps source-ctxt.simps take-map drop-map* **using** *iq* **by** *auto*
with *ctxt* **have** $p = i\#p'$
by (*metis hole-pos-ctxt-of-pos-term p*)
with *p* **show** *?thesis*
by *auto*
qed

lemma *simple-pterm-match*:
assumes $source\ A = t \cdot \sigma$
and *linear-term t*
and $A \cdot \tau 1 = to\text{-}pterm\ t \cdot \tau 2$
shows *matches A* (*to-pterm t*)
using *assms* **proof**(*induct t arbitrary*: *A*)
case (*Var x*)
then show *?case*
using *matches-iff* **by** *force*
next
case (*Fun f ts*)
from *Fun*(*2*,*4*) **show** *?case* **proof**(*cases A*)

case (*Pfun g As*)
with *Fun*(*2*) **have** *f*:*f* = *g* **by** *simp*
from *Fun*(*2*) **have** *l*:*length ts* = *length As*
unfolding *Pfun source.simps f eval-term.simps* **by** (*simp add*: *map-equality-iff*)

{**fix** *i* **assume** *i*:*i* < *length ts*
with *Fun*(*2*) **have** *source* (*As* ! *i*) = *ts* ! *i* · *σ*
unfolding *Pfun source.simps f eval-term.simps* **by** (*simp add*: *map-equality-iff*)
moreover from *i Fun*(*4*) **have** *As* ! *i* · *τ1* = *to-pterm* (*ts* ! *i*) · *τ2*
unfolding *Pfun f to-pterm.simps eval-term.simps* **using** *l map-nth-conv* **by** *fastforce*
ultimately have *matches* (*As*!*i*) (*to-pterm* (*ts*!*i*))
using *Fun*(*1*)[*of ts*!*i As*!*i*] *l i Fun*(*3*) **by** *force*
then have ∃ *σ*. *As*!*i* = (*to-pterm* (*ts*!*i*)) · *σ*
by (*metis matches-iff*)
}**note** *IH*=*this*
from *Fun*(*3*) **have** *lin*:*linear-term* (*to-pterm* (*Fun f ts*))
using *to-pterm-linear* **by** *blast*
from *linear-term-obtain-subst*[*OF lin*[*unfolded to-pterm.simps*]] **show** *?thesis*
unfolding *Pfun f* **by** (*smt* (*verit*, *del-insts*) *IH l length-map matches-iff nth-map to-pterm.simps*(*2*))
qed *simp-all*
qed

2.9 Proof Terms Represent Multi-Steps

context *var-rhs-subset-lhs*
begin
lemma *mstep-to-pterm*:
assumes (*s*, *t*) ∈ *mstep R*
shows ∃ *A*. *A* ∈ *wf-pterm R* ∧ *source A* = *s* ∧ *target A* = *t*
using *assms*(*1*) **proof**(*induct*)
case (*Var x*)
then show *?case*
by (*meson source.simps*(*1*) *target.simps*(*1*) *wf-pterm.intros*(*1*))
next
case (*args f n ss ts*)
then have ∀ *i*∈*set* [*0*..<*n*]. ∃ *a*. *a* ∈ *wf-pterm R* ∧ *source a* = *ss* ! *i* ∧ *target a* = *ts* ! *i*
by *simp*
then obtain *As* **where** *as*:*length As* = *n* ∧ (∀ *i* < *n*. (*As*!*i*) ∈ *wf-pterm R* ∧ *source* (*As*!*i*) = *ss* ! *i* ∧ *target* (*As*!*i*) = *ts* ! *i*)
using *obtain-list-with-property*[**where** *P*=λ*a i*. *a* ∈ *wf-pterm R* ∧ *source a* = *ss*!*i* ∧ *target a* = *ts*!*i* **and** *xs*=[*0*..<*n*]]
by (*metis add.left-neutral diff-zero length-upt nth-upt set-upt*)
with *args*(*1*) **have** *source* (*Pfun f As*) = *Fun f ss*
unfolding *source.simps* **by** (*simp add*: *map-nth-eq-conv*)
moreover from *as args*(*2*) **have** *target* (*Pfun f As*) = *Fun f ts*
unfolding *target.simps* **by** (*simp add*: *map-nth-eq-conv*)

ultimately show *?case*
using *as* **by** (*metis in-set-idx wf-pterm.intros*(*2*))
next
case (*rule l r σ τ*)
let *?α* =(*l* → *r*)
have *set* (*vars-distinct l*) = *vars-term l*
by *simp*
with *rule*(*2*) **obtain** *As* **where** *as*:*length As* = *length* (*vars-distinct l*) ∧
(∀ *i* < *length* (*vars-distinct l*). (*As*!*i*) ∈ *wf-pterm R* ∧
source (*As*!*i*) = *σ* ((*vars-distinct l*) ! *i*) ∧ *target* (*As*!*i*) = *τ* ((*vars-distinct l*) ! *i*))
using *obtain-list-with-property*[**where** *P*=λ*a x*. *a* ∈ *wf-pterm R* ∧ *source a* = *σ x* ∧ *target a* = *τ x*] **by** *blast*
with *rule*(*1*) **have** *well*:*Prule ?α As* ∈ *wf-pterm R*
by (*metis in-set-idx prule.sel*(*1*) *prule.sel*(*2*) *wf-pterm.simps*)
from *as* **have** ∀ *x* ∈ *vars-term l*. (⟨*map source As*⟩$_?$*α*) *x* = *σ x*
by (*smt* (*z3*) *apply-lhs-subst-var-rule in-set-idx length-map map-nth-conv prule.sel*(*1*) *set-vars-term-list vars-term-list-vars-distinct*)
then have *s*:*source* (*Prule ?α As*) = *l* · *σ*
by (*simp add*: *term-subst-eq-conv*)
from *as varcond* **have** ∀ *x* ∈ *vars-term r*. (⟨*map target As*⟩$_?$*α*) *x* = *τ x*
by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule fst-conv in-set-conv-nth length-map nth-map prule.sel*(*1*)
rule.hyps(*1*) *set-vars-term-list snd-conv split-beta subsetD vars-term-list-vars-distinct*)

then have *target* (*Prule ?α As*) = *r* · *τ*
by (*simp add*: *term-subst-eq-conv*)
with *well s* **show** *?case*
by *blast*
qed
end

lemma *pterm-to-mstep*:
assumes *A* ∈ *wf-pterm R*
shows (*source A*, *target A*) ∈ *mstep R*
using *assms* **proof**(*induct*)
case (*2 As f*)
then show *?case*
by (*simp add*: *mstep.args*)
next
case (*3 α As*)
then have ∀ *x*∈*vars-term* (*lhs α*). ((⟨*map source As*⟩$_α$) *x*, (⟨*map target As*⟩$_α$) *x*) ∈ *mstep R*
by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule comp-def in-set-idx length-map map-nth-conv nth-mem set-remdups set-rev set-vars-term-list*)
with *3*(*1*) **show** *?case*
by (*simp add*: *mstep.rule*)
qed *simp*

lemma *co-init-prule*:
assumes *co-initial* (*Prule* α *As*) (*Prule* α *Bs*)
and *Prule* α *As* $\in$ *wf-pterm R* **and** *Prule* α *Bs* $\in$ *wf-pterm R*
shows $\forall$ *i*<*length As*. *co-initial* (*As*!*i*) (*Bs*!*i*)
proof−
from *assms* **have** *l1*:*length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
from *assms* **have** *l2*:*length Bs* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
{fix *i* **assume** *i*:*i* < *length As* **and** *co*:¬ (*co-initial* (*As*!*i*) (*Bs*!*i*))
then have ($\langle$*map source As*$\rangle_\alpha$) ((*var-rule* α)!*i*) ≠ ($\langle$*map source Bs*$\rangle_\alpha$) ((*var-rule* α)!*i*)
by (*metis l1 l2 length-map lhs-subst-var-i nth-map*)
with *assms*(*1*) **have** *False* **unfolding** *source.simps*
by (*smt* (*z3*) *comp-apply i l1 nth-mem set-remdups set-rev set-vars-term-list term-subst-eq-rev*)
} then show *?thesis*
by *blast*
qed

3 Operations on Proof Terms

The operations residual, deletion, and join on proof terms all fulfill *A* ⋆ (*source A*) = *A* which implies several useful results.

locale *op-proof-term* = *left-lin-no-var-lhs* +
fixes *f* :: (('*a*, '*b*) *prule* + '*a*, '*b*) *Term.term* ⇒ (('*a*, '*b*) *prule* + '*a*, '*b*) *Term.term* ⇒ (('*a*, '*b*) *prule* + '*a*, '*b*) *Term.term option*
assumes *f-src*: *A* ∈ *wf-pterm R* ⟹ *f A* (*to-pterm* (*source A*)) = *Some A*
and *f-pfun*:*f* (*Pfun g As*)(*Pfun g Bs*) = (*if length As* = *length Bs then*
(*case those* (*map2 f As Bs*) *of*
Some xs ⇒ *Some* (*Pfun g xs*)
| *None* ⇒ *None*) *else None*)
and *f-prule*:*f* (*Prule* α *As*) (*Pfun g Bs*) = (*case match* (*Pfun g Bs*) (*to-pterm* (*lhs* α)) *of*
None ⇒ *None*
| *Some* σ ⇒
(*case those* (*map2 f As* (*map* σ (*var-rule* α))) *of*
Some xs ⇒ *Some* (*Prule* α *xs*)
| *None* ⇒ *None*))
begin

notation
f ('(⋆')) **and**
f ((- ⋆ -) [*51*, *51*] *50*)

lemma *apply-f-ctxt*:
assumes *C* ∈ *wf-pterm-ctxt R*
and *A* ⋆ *B* = *Some D*

```
  shows C⟨A⟩ ⋆ (to-pterm-ctxt (source-ctxt C))⟨B⟩ = Some (C⟨D⟩)
  using assms proof(induct C rule:pterm-ctxt-induct)
  case (Cfun f ss1 C ss2)
    have l:length ((map (to-pterm ∘ source) ss1) @ (to-pterm-ctxt (source-ctxt
C))⟨A⟩ # (map (to-pterm ∘ source) ss2))
          = length (ss1 @ C⟨B⟩ # ss2) by auto
   from Cfun(2) have well1:∀ i < length ss1. (ss1!i) ∈ wf-pterm R by auto
   from Cfun(2) have well2:∀ i < length ss2. (ss2!i) ∈ wf-pterm R by auto
   from Cfun have fC:C⟨A⟩ ⋆ (to-pterm-ctxt (source-ctxt C))⟨B⟩ = Some (C⟨D⟩)

     by auto
   from well1 have f1:∀ i < length ss1. ((map2 (⋆) ss1 (map (to-pterm ∘ source)
ss1))!i = Some (ss1!i))
     using f-src to-pterm-empty by fastforce
   from well2 have f2:∀ i < length ss2. ((map2 (⋆) ss2 (map (to-pterm ∘ source)
ss2))!i = Some (ss2!i))
     using f-src to-pterm-empty by fastforce
   {fix i assume i:i < (length ss1) + (length ss2) +1
     have (map2 (⋆) (ss1 @ (C⟨A⟩ # ss2))
           (map (to-pterm ∘ source) ss1 @ ((to-pterm-ctxt (source-ctxt C))⟨B⟩ #
map (to-pterm ∘ source) ss2)))!i
          = Some ((ss1 @ C⟨D⟩ # ss2)!i)
     proof−
       consider i < length ss1 | i = length ss1 | i > length ss1
         using nat-neq-iff by blast
       then show ?thesis proof(cases)
         case 1
         then show ?thesis using f1
           by (simp add: append-Cons-nth-left)
       next
         case 2
         then show ?thesis using fC
           by (simp add: append-Cons-nth-middle)
       next
         case 3
         with i have l:(map (to-pterm ∘ source) ss1 @ (to-pterm-ctxt (source-ctxt
C))⟨B⟩ # map (to-pterm ∘ source) ss2)!i
                     = (map (to-pterm ∘ source) ss2)!(i−(length ss1 + 1))
           by (metis add.commute length-map less-SucI not-less-eq nth-append-Cons
plus-1-eq-Suc)
        from 3 i have r:(ss1 @ (C⟨to-pterm (source B)⟩ # ss2))!i = ss2!(i−(length
ss1 + 1))
         by (metis add.commute less-SucI not-less-eq nth-append-Cons plus-1-eq-Suc)

         from l r 3 show ?thesis using f2
        by (smt One-nat-def add.right-neutral add-Suc add-Suc-right add-diff-inverse-nat
add-less-cancel-left append-Cons-nth-right i length-append length-map length-zip list.size(4)
min-less-iff-conj not-less-eq nth-map nth-zip)
       qed
```

```
    qed
  }
  with l have those ((map2 (⋆) (ss1 @ (C⟨A⟩ # ss2))
          (map (to-pterm ∘ source) ss1 @ ((to-pterm-ctxt (source-ctxt C))⟨B⟩ #
map (to-pterm ∘ source) ss2))))
         = Some (ss1 @ C⟨D⟩ # ss2) by (simp add: those-some)
  with l show ?case using f-pfun by simp
 next
  case (Crule α ss1 C ss2)
  from Crule(2) have alpha:to-rule α ∈ R
    using wf-pterm-ctxt.cases by auto
  then have linear:linear-term (lhs α)
    using left-lin left-linear-trs-def by fastforce
  then have linear':linear-term (to-pterm (lhs α))
    using to-pterm-linear by blast
   have l1:length ((map (to-pterm ∘ source) ss1) @ (to-pterm-ctxt (source-ctxt
C))⟨A⟩ # (map (to-pterm ∘ source) ss2))
        = length (ss1 @ C⟨B⟩ # ss2) by auto
  from Crule(2) have l2:length (ss1 @ C⟨B⟩ # ss2) = length (var-rule α)
    using wf-pterm-ctxt.simps by fastforce
  from Crule(2) have well1:∀ i < length ss1. (ss1!i) ∈ wf-pterm R by auto
  from Crule(2) have well2:∀ i < length ss2. (ss2!i) ∈ wf-pterm R by auto
  from Crule have fC:C⟨A⟩ ⋆ (to-pterm-ctxt (source-ctxt C))⟨B⟩ = Some (C⟨D⟩)

    by auto
  from well1 have f1:∀ i < length ss1. ((map2 (⋆) ss1 (map (to-pterm ∘ source)
ss1))!i = Some (ss1!i))
    using f-src to-pterm-empty by fastforce
  from well2 have f2:∀ i < length ss2. ((map2 (⋆) ss2 (map (to-pterm ∘ source)
ss2))!i = Some (ss2!i))
    using f-src to-pterm-empty by fastforce
  {fix i assume i:i < (length ss1) + (length ss2) +1
     have (map2 (⋆) (ss1 @ (C⟨A⟩ # ss2)) (map (to-pterm ∘ source) ss1 @
((to-pterm-ctxt (source-ctxt C))⟨B⟩ #
        (map (to-pterm ∘ source) ss2))) )!i = Some ((ss1 @ C⟨D⟩ # ss2)!i)
    proof−
     consider i < length ss1 | i = length ss1 | i > length ss1
       using nat-neq-iff by blast
     then show ?thesis proof(cases)
       case 1
       then show ?thesis using f1
         by (simp add: append-Cons-nth-left)
     next
       case 2
       then show ?thesis using fC
         by (simp add: append-Cons-nth-middle)
     next
       case 3
       with i have l:(map (to-pterm ∘ source) ss1 @ (to-pterm-ctxt (source-ctxt
```

C))⟨*B*⟩ # *map* (*to-pterm* ∘ *source*) *ss2*)!*i*
= (*map* (*to-pterm* ∘ *source*) *ss2*)!(*i*−(*length ss1* + *1*))
by (*metis add.commute length-map less-SucI not-less-eq nth-append-Cons plus-1-eq-Suc*)
from *3 i* **have** *r*:(*ss1* @ (*C*⟨*to-pterm* (*source B*)⟩ # *ss2*))!*i* = *ss2*!(*i*−(*length ss1* + *1*))
by (*metis add.commute less-SucI not-less-eq nth-append-Cons plus-1-eq-Suc*)

from *l r 3* **show** *?thesis* **using** *f2*
by (*smt One-nat-def add.right-neutral add-Suc add-Suc-right add-diff-inverse-nat add-less-cancel-left append-Cons-nth-right i length-append length-map length-zip list.size*(*4*) *min-less-iff-conj not-less-eq nth-map nth-zip*)
qed
qed
}
with *l1* **have** *IH*:*those* (*map2* (⋆) (*ss1* @ (*C*⟨*A*⟩ # *ss2*)) (*map* (*to-pterm* ∘ *source*) *ss1* @ ((*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩ #
(*map* (*to-pterm* ∘ *source*) *ss2*)))) = *Some* (*ss1* @ *C*⟨*D*⟩ # *ss2*) **by** (*simp add*: *those-some*)
let *?p* = (*var-poss-list* (*lhs* α) ! *length ss1*)
let *?x* = *vars-term-list* (*lhs* α) ! *length ss1*
let *?σ* = ⟨*map source* (*ss1* @ *Var* (*vars-term-list* (*lhs* α) ! *length ss1*) # *ss2*)⟩$_\alpha$
from *l2 linear* **have** *l3*:*length ss1* < *length* (*var-poss-list* (*lhs* α))
by (*metis* (*no-types*, *lifting*) *add-Suc-right append-Cons-nth-left le-imp-less-Suc length-append length-var-poss-list linear-term-var-vars-term-list linorder-neqE-nat list.size*(*3*) *list.size*(*4*) *not-add-less1 nth-equalityI self-append-conv zero-order*(*1*))
then have *?p* ∈ *poss* (*lhs* α)
using *nth-mem var-poss-imp-poss var-poss-list-sound* **by** *blast*
then have *ctxt*:(*to-pterm-ctxt* (*source-ctxt* (*Crule* α *ss1 C ss2*)))⟨*B*⟩ =
(*ctxt-of-pos-term ?p* (*to-pterm* (*lhs* α) · (*to-pterm* ∘ *?σ*)))⟨(*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩⟩
unfolding *source-ctxt.simps intp-actxt.simps Let-def ctxt-ctxt-compose to-pterm-ctxt-comp*

using *to-pterm-ctxt-at-pos*[**where** *?p*=*?p* **and** *?s*=*lhs* α · *?σ*] **by** (*simp add*: *to-pterm-subst*)
from *l3* **have** *l4*:*length ss1* < *length* (*vars-term-list* (*to-pterm* (*lhs* α)))
by (*metis length-var-poss-list vars-to-pterm*)
have (*to-pterm-ctxt* (*source-ctxt* (*Crule* α *ss1 C ss2*)))⟨*B*⟩ =
to-pterm (*lhs* α) · ((*to-pterm* ∘ *?σ*)(*?x* := (*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩))
unfolding *ctxt* **using** *ctxt-apply-term-subst*[**where** *?p*=*?p* **and** *?t*=*to-pterm* (*lhs* α) **and** *?i*=*length ss1* **and** *?s*=(*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩ **and** *?σ*=(*to-pterm* ∘ *?σ*)]
linear′ l4 var-poss-list-to-pterm vars-to-pterm **by** *metis*
then obtain τ **where** τ:*match* (*to-pterm-ctxt* (*source-ctxt* (*Crule* α *ss1 C ss2*)))⟨*B*⟩ (*to-pterm* (*lhs* α)) = *Some* τ
unfolding *ctxt* **using** *ctxt-apply-term-subst linear′ match-complete′ option.distinct*(*1*) **by** *force*
have *varr*:(*var-rule* α) = *vars-term-list* (*to-pterm* (*lhs* α))

using *linear linear-term-var-vars-term-list* **unfolding** *vars-to-pterm* **by** *force*
have (*map* (*to-pterm* ∘ *?σ*) (*vars-term-list* (*to-pterm* (*lhs* α)))) = *map* (*to-pterm* ∘ *source*) (*ss1* @ *Var* (*vars-term-list* (*lhs* α) ! *length ss1*) # *ss2*)
using *apply-lhs-subst-var-rule l2* **unfolding** *varr*[*symmetric*] **by** *force*
then have (*map* (*to-pterm* ∘ *?σ*) (*vars-term-list* (*to-pterm* (*lhs* α))))[*length ss1* := (*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩] =
(*map* (*to-pterm* ∘ *source*) *ss1* @ (*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩ # (*map* (*to-pterm* ∘ *source*) *ss2*))
by (*metis* (*no-types*, *lifting*) *length-map list.simps*(*9*) *list-update-length map-append*)

with τ **have** *map-tau*:*map* τ (*var-rule* α) = (*map* (*to-pterm* ∘ *source*) *ss1* @ (*to-pterm-ctxt* (*source-ctxt C*))⟨*B*⟩ #
(*map* (*to-pterm* ∘ *source*) *ss2*))
using *match-lhs-context*[**where** *?t*=*to-pterm* (*lhs* α) **and** *?τ*=τ **and** *?σ*=(*to-pterm* ∘ *?σ*)]
l4 var-poss-list-to-pterm linear′ ctxt varr **by** *metis*
from *alpha no-var-lhs* **obtain** *f ts* **where** *f*:*lhs* α = *Fun f ts*
by *blast*
have [] ∉ *var-poss* (*lhs* α)
unfolding *f var-poss.simps* **by** *force*
then obtain *i q* **where** *iq*:*?p* = *i* # *q* **using** *l3*
by (*metis in-set-conv-nth subt-at.elims var-poss-list-sound*)
then obtain *ts′* **where** *root-not-rule*:(*to-pterm-ctxt* (*source-ctxt* (*Crule* α *ss1 C ss2*)))⟨*B*⟩ = *Pfun f ts′*
unfolding *ctxt iq* **unfolding** *f* **by** *simp*
then show *?case*
using τ *f-prule map-tau IH* **by** *force*
qed *simp*

end

end
theory *Residual-Join-Deletion*

imports
Proof-Terms
Linear-Matching
begin

3.1 Residuals

Auxiliary lemma in preparation of termination simp rule.

lemma *match-vars-term-size*:
assumes *match s t* = *Some* σ
and *x* ∈ *vars-term t*
shows *size* (σ *x*) ≤ *size s*
using *assms vars-term-size* **by** (*metis match-matches*)

lemma [*termination-simp*]:

```
  assumes match (Fun f ss) (to-pterm l) = Some σ
    and ∗: (s, t) ∈ set (zip (map σ (vars-distinct l)) ts)
  shows size s ≤ Suc (size-list size ss)
proof −
  from ∗ have s ∈ set (map σ (vars-distinct l)) by (blast elim: in-set-zipE)
  then obtain x where [simp]: s = σ x
    and x: x ∈ vars-term (to-pterm l) by (induct l) auto
  from match-vars-term-size [OF assms(1)  x]
    show ?thesis by simp
  qed
```

Additional simp rule because we allow variable left-hand sides of rewrite rules at this point. Then *Var x / α* and *α / Var x* are also possible when evaluating residuals. This might become important when we want to introduce the error rule for residuals of composed proof terms.

```
lemma [termination-simp]:
  assumes match (Var x) (to-pterm l) = Some σ
    and (a, b) ∈ set (zip (map σ (vars-distinct l)) ts)
  shows size a = 1
proof−
  from assms(1) have ∗:(to-pterm l) · σ = Var x by (simp add: match-matches)
  then obtain y where y:l = Var y by (metis subst-apply-eq-Var term.distinct(1)
to-pterm.elims)
  with ∗ have ∗∗:σ y = Var x by simp
  from y have vars-distinct l = [y] by (simp add: vars-term-list.simps(1))
  with assms(2) y have a = Var x by (simp add: ∗∗ in-set-zip)
  then show ?thesis by simp
qed

fun residual :: ('f, 'v) pterm ⇒ ('f, 'v) pterm ⇒ ('f, 'v) pterm option (infixr re
70)
  where
  Var x re Var y =
    (if x = y then Some (Var x) else None)
| Pfun f As re Pfun g Bs =
    (if (f = g ∧ length As = length Bs) then
      (case those (map2 residual As Bs) of
        Some xs ⇒ Some (Pfun f xs)
      | None ⇒ None)
    else None)
| Prule α As re Prule β Bs =
    (if α = β then
      (case those (map2 residual As Bs) of
        Some xs ⇒ Some ((to-pterm (rhs α)) · ⟨xs⟩α)
      | None ⇒ None)
    else None)
| Prule α As re B =
    (case match B (to-pterm (lhs α)) of
      None ⇒ None
```

```
    | Some σ ⇒
      (case those (map2 residual As (map σ (var-rule α))) of
        Some xs ⇒ Some (Prule α xs)
      | None ⇒ None))
| A re Prule α Bs =
    (case match A (to-pterm (lhs α)) of
      None ⇒ None
    | Some σ ⇒
      (case those (map2 residual (map σ (var-rule α)) Bs) of
        Some xs ⇒ Some ((to-pterm (rhs α)) · ⟨xs⟩α)
      | None ⇒ None))
| A re B = None
```

Since the interesting proofs about residuals always follow the same pattern of induction on the definition, we introduce the following 6 lemmas corresponding to the step cases.

```
lemma residual-fun-fun:
  assumes (Pfun f As) re (Pfun g Bs) = Some C
  shows f = g ∧ length As = length Bs ∧
        (∃ Cs. C = Pfun f Cs ∧
        length Cs = length As ∧
        (∀ i < length As. As!i re Bs!i = Some (Cs!i)))
proof−
  have ∗:f = g ∧ length As = length Bs
    using assms residual.simps(2) by (metis option.simps(3))
  then obtain Cs where Cs:those (map2 (re) As Bs) = Some Cs
    using assms residual.simps(2) option.simps(3) option.simps(4) by fastforce
  hence ∀ i < length As. As!i re Bs!i = Some (Cs!i)
    using ∗ those-some2 by fastforce
  with ∗ Cs assms(1) show ?thesis
    using length-those by fastforce
qed

lemma residual-rule-rule:
  assumes (Prule α As) re (Prule β Bs) = Some C
          (Prule α As) ∈ wf-pterm R
          (Prule β Bs) ∈ wf-pterm S
  shows α = β ∧ length As = length Bs ∧
        (∃ Cs. C = to-pterm (rhs α) · ⟨Cs⟩α ∧
        length Cs = length As ∧
        (∀ i < length As. As!i re Bs!i = Some (Cs!i)))
proof−
  have α = β
    using assms(1) residual.simps(3) by (metis option.simps(3))
  with assms(2,3) have l: length As = length Bs
    using length-args-well-Prule by blast
  from ‹α = β› obtain Cs where Cs:those (map2 (re) As Bs) = Some Cs
    using assms by fastforce
  hence ∀ i < length As. As!i re Bs!i = Some (Cs!i)
```

```
    using l those-some2 by fastforce
  with ‹α = β› l Cs assms(1) show ?thesis
    using length-those by fastforce
qed

lemma residual-rule-var:
  assumes (Prule α As) re (Var x) = Some C
          (Prule α As) ∈ wf-pterm R
  shows ∃σ. match (Var x) (to-pterm (lhs α)) = Some σ ∧
        (∃ Cs. C = Prule α Cs ∧
        length Cs = length As ∧
        (∀ i < length As. As!i re (σ (var-rule α ! i)) = Some (Cs!i)))
proof−
  from assms(2) have l:length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  obtain σ where σ:match (Var x) (to-pterm (lhs α)) = Some σ
    using assms(1) by fastforce
  then obtain Cs where Cs:those (map2 residual As (map σ (var-rule α))) =
Some Cs
    using assms(1) by fastforce
  with l have l2:length Cs = length As
    using length-those by fastforce
  from Cs have ∀ i < length As. As!i re (σ (var-rule α ! i)) = Some (Cs!i)
    using l those-some2 by fastforce
  with σ Cs assms(1) l2 show ?thesis by simp
qed

lemma residual-rule-fun:
  assumes (Prule α As) re (Pfun f Bs) = Some C
          (Prule α As) ∈ wf-pterm R
  shows ∃σ. match (Pfun f Bs) (to-pterm (lhs α)) = Some σ ∧
        (∃ Cs. C = Prule α Cs ∧
        length Cs = length As ∧
        (∀ i < length As. As!i re (σ (var-rule α ! i)) = Some (Cs!i)))
proof−
  from assms(2) have l:length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  obtain σ where σ:match (Pfun f Bs) (to-pterm (lhs α)) = Some σ
    using assms(1) by fastforce
  then obtain Cs where Cs:those (map2 residual As (map σ (var-rule α))) =
Some Cs
    using assms(1) by fastforce
  with l have l2:length Cs = length As
    using length-those by fastforce
  from Cs have ∀ i < length As. As!i re (σ (var-rule α ! i)) = Some (Cs!i)
    using l those-some2 by fastforce
  with σ Cs assms(1) l2 show ?thesis by auto
qed
```

lemma *residual-var-rule*:
assumes (*Var x*) *re* (*Prule α As*) = *Some C*
(*Prule α As*) ∈ *wf-pterm R*
shows ∃ *σ*. *match* (*Var x*) (*to-pterm* (*lhs α*)) = *Some σ* ∧
(∃ *Cs*. *C* = (*to-pterm* (*rhs α*)) · ⟨*Cs*⟩$_\alpha$ ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. (*σ* (*var-rule α* ! *i*) *re As*!*i* = *Some* (*Cs*!*i*)))
proof−
from *assms*(*2*) **have** *l*:*length As* = *length* (*var-rule α*)
using *wf-pterm.simps* **by** *fastforce*
obtain *σ* **where** *σ*:*match* (*Var x*) (*to-pterm* (*lhs α*)) = *Some σ*
using *assms*(*1*) **by** *fastforce*
then obtain *Cs* **where** *Cs*:*those* (*map2 residual* (*map σ* (*var-rule α*)) *As*) = *Some Cs*
using *assms*(*1*) **by** *fastforce*
with *l* **have** *l2*:*length Cs* = *length As*
using *length-those* **by** *fastforce*
from *Cs* **have** ∀ *i* < *length As*. (*σ* (*var-rule α* ! *i*)) *re As*!*i* = *Some* (*Cs*!*i*)
using *l those-some2* **by** *fastforce*
with *σ Cs assms*(*1*) *l2* **show** *?thesis* **by** *auto*
qed

lemma *residual-fun-rule*:
assumes (*Pfun f Bs*) *re* (*Prule α As*) = *Some C*
(*Prule α As*) ∈ *wf-pterm R*
shows ∃ *σ*. *match* (*Pfun f Bs*) (*to-pterm* (*lhs α*)) = *Some σ* ∧
(∃ *Cs*. *C* = (*to-pterm* (*rhs α*)) · ⟨*Cs*⟩$_\alpha$ ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. (*σ* (*var-rule α* ! *i*)) *re As*!*i* = *Some* (*Cs*!*i*)))
proof−
from *assms*(*2*) **have** *l*:*length As* = *length* (*var-rule α*)
using *wf-pterm.simps* **by** *fastforce*
obtain *σ* **where** *σ*:*match* (*Pfun f Bs*) (*to-pterm* (*lhs α*)) = *Some σ*
using *assms*(*1*) **by** *fastforce*
then obtain *Cs* **where** *Cs*:*those* (*map2 residual* (*map σ* (*var-rule α*)) *As*) = *Some Cs*
using *assms*(*1*) **by** *fastforce*
with *l* **have** *l2*:*length Cs* = *length As*
using *length-those* **by** *fastforce*
with *Cs* **have** ∀ *i* < *length As*. (*σ* (*var-rule α* ! *i*)) *re As*!*i* = *Some* (*Cs*!*i*)
using *l those-some2* **by** *fastforce*
with *σ Cs assms*(*1*) *l2* **show** *?thesis* **by** *auto*
qed

$t \;/\; A = tgt(A)$

lemma *res-empty1*:
assumes *is-empty-step t co-initial A t A* ∈ *wf-pterm R*
shows *t re A* = *Some* (*to-pterm* (*target A*))
proof −

from *assms*(*1*,*2*) **have** t = *to-pterm* (*source A*)
by (*simp add*: *empty-coinitial*)
then show *?thesis* **using** *assms*(*3*) **proof** (*induction A arbitrary*: t)
case (*Var x*)
then show *?case* **by** *simp*
next
case (*Pfun f As*)
let *?ts* = (*map* (*to-pterm* ∘ *source*) *As*)
from *Pfun*(*3*) **have** ∀ a ∈ *set As*. a ∈ *wf-pterm R* **by** *blast*
with *Pfun*(*1*) **have** *those* (*map2 residual ?ts As*) = *Some* (*map* (*to-pterm* ∘ *target*) *As*) **by** (*simp add*:*those-some*)
then show *?case* **unfolding** *Pfun*(*2*) **by** *simp*
next
case (*Prule* α *As*)
let *?ts* = (*map* (*to-pterm* ∘ *source*) *As*)
from *Prule*(*3*) **have** *l*:*length ?ts* = *length* (*var-rule* α) **using** *wf-pterm.simps* **by** *fastforce*
moreover from *Prule*(*3*) **have** *well*:∀ a ∈ *set As*. a ∈ *wf-pterm R* **by** *blast*
from *Prule*(*1*) **have** *args*:*those* (*map2 residual ?ts As*) = *Some* (*map* (*to-pterm* ∘ *target*) *As*) **using** *well* **by** (*simp add*:*those-some*)
from *Prule*(*2*) **have** t:t = (*to-pterm* (*lhs* α)) · $\langle$*?ts*$\rangle_\alpha$ **by** (*simp add*: *to-pterm-subst*)

then obtain σ **where** σ:
match t (*to-pterm* (*lhs* α)) = *Some* σ
(∀ x∈ *set* (*var-rule* α). ($\langle$*?ts*$\rangle_\alpha$) x = σ x)
using *lhs-subst-trivial* **by** *blast*
from σ(*2*) *l* **have** *ts*:*map* σ (*var-rule* α) = *?ts* **by** (*smt apply-lhs-subst-var-rule map-eq-conv*)
from *Prule*(*1*) **have** *those* (*map2 residual ?ts As*) = *Some* (*map* (*to-pterm* ∘ *target*) *As*) **using** *well* **by** (*simp add*:*those-some*)
with *ts* **have** *args*:*those* (*map2 residual* (*map* σ (*var-rule* α)) *As*) = *Some* (*map* (*to-pterm* ∘ *target*) *As*) **by** *simp*
show *?case* **proof** (*cases t rule*:*source.cases*)
case (*1 x*)
then show *?thesis* **using** *args* σ(*1*) **by** (*simp add*: *to-pterm-subst*)
next
case (*2 f As*)
then show *?thesis* **using** *args* σ(*1*) **by** (*simp add*: *to-pterm-subst*)
next
case (*3* α *As*)
then show *?thesis* **using** *Prule*(*2*) **by** (*metis is-empty-step.simps*(*3*) *to-pterm-empty*)
qed
qed
qed

A / t = *A*

lemma *res-empty2*:
assumes *A* ∈ *wf-pterm R*
shows *A re* (*to-pterm* (*source A*)) = *Some A*

using *assms* **proof** (*induction A*)
case (*2 As f*)
then have *those* (*map2 residual As* (*map* (*to-pterm* ∘ *source*) *As*)) = *Some As*
by (*simp add*:*those-some*)
then show *?case* **by** *simp*
next
case (*3* α *As*)
then have σ: *match* (*to-pterm* (*lhs* α · ⟨*map source As*⟩$_\alpha$)) (*to-pterm* (*lhs* α)) = *Some* (⟨*map* (*to-pterm* ∘ *source*) *As*⟩$_\alpha$)
by (*metis* (*no-types*, *lifting*) *fun-mk-subst lhs-subst-trivial map-map to-pterm.simps*(*1*) *to-pterm-subst*)
from *3* **have** *those* (*map2 residual As* (*map* (*to-pterm* ∘ *source*) *As*)) = *Some As*
by (*simp add*:*those-some*)
then have *args*:*those* (*map2 residual As* (*map* (⟨*map* (*to-pterm* ∘ *source*) *As*⟩$_\alpha$) (*var-rule* α))) = *Some As*
by (*metis 3.hyps*(*2*) *apply-lhs-subst-var-rule length-map*)
show *?case* **proof**(*cases source* (*Prule* α *As*))
case (*Var x*)
then show *?thesis*
using σ *residual.simps*(*4*)[*of* α *As x*] *args* **by** *auto*
next
case (*Fun f ts*)
then show *?thesis*
using σ *residual.simps*(*5*)[*of* α *As f*] *args* **by** *auto*
qed
qed *simp*

A / *A* = *tgt*(*A*)

lemma *res-same*: *A re A* = *Some* (*to-pterm* (*target A*))
proof(*induction A*)
case (*Var x*)
then show *?case* **by** *simp*
next
case (*Pfun f As*)
then have *list-all* (λ*x*. *x* ≠ *None*) (*map2 residual As As*) **by** (*simp add*: *list-all-length*)
then obtain *xs* **where** *xs*:*those* (*map2 residual As As*) = *Some xs* **using** *those-not-none-xs*
by *fastforce*
then have *l*:*length As* = *length xs* **using** *length-those* **by** *fastforce*
from *xs* **have** *IH*:*i* < *length As* ⟹ *xs*!*i* = *to-pterm* (*target* (*As*!*i*)) **for** *i* **using** *Pfun those-some2* **by** *force*
from *IH l* **have** *map* (*to-pterm*∘*target*) *As* = *xs* **by** (*simp add*: *map-nth-eq-conv*)
then have *to-pterm* (*target* (*Pfun f As*)) = *Pfun f xs* **by** *simp*
then show *?case* **using** *xs* **by** *simp*
next
case (*Prule* α *As*)
then have *list-all* (λ*x*. *x* ≠ *None*) (*map2 residual As As*) **by** (*simp add*: *list-all-length*)
then obtain *xs* **where** *xs*:*those* (*map2 residual As As*) = *Some xs* **using** *those-not-none-xs*
by *fastforce*

then have *l*:*length As* = *length xs* **using** *length-those* **by** *fastforce*
from *xs* **have** *IH*:*i* < *length As* ⟹ *xs*!*i* = *to-pterm* (*target* (*As*!*i*)) **for** *i* **using** *Prule those-some2* **by** *force*
from *IH l* **have** ∗:*map* (*to-pterm*∘*target*) *As* = *xs* **by** (*simp add*: *map-nth-eq-conv*)
have *to-pterm* (*target* (*Prule* α *As*)) = *to-pterm* (*rhs* α · ⟨*map target As*⟩$_\alpha$) **by** *simp*
also have ... = (*to-pterm* (*rhs* α)) · (*to-pterm* ∘ ⟨*map target As*⟩$_\alpha$) **by** (*simp add*: *to-pterm-subst*)
also have ... = (*to-pterm* (*rhs* α)) · ⟨*xs*⟩$_\alpha$ **using** ∗ **by** *simp*
finally show *?case* **using** *xs* **by** *simp*
qed

lemma *residual-src-tgt*:
assumes *A re B* = *Some C A* ∈ *wf-pterm R B* ∈ *wf-pterm S*
shows *source C* = *target B*
using *assms* **proof**(*induction A B arbitrary*: *C rule*: *residual.induct*)
case (*1 x y*)
then show *?case*
by (*metis option.distinct*(*1*) *option.sel residual.simps*(*1*) *source.simps*(*1*) *target.simps*(*1*))
next
case (*2 f As g Bs*)
then obtain *Cs* **where** ∗:*f* = *g* ∧ *length As* = *length Bs* ∧
C = *Pfun f Cs* ∧ *length Cs* = *length As* ∧
(∀ *i*<*length As*. *As* ! *i re Bs* ! *i* = *Some* (*Cs* ! *i*))
by (*meson residual-fun-fun*)
then have *length* (*zip As Bs*) = *length As* **by** *simp*
moreover from *2*(*3*) **have** ∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R* **by** *blast*
moreover from *2*(*4*) **have** ∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm S* **by** *blast*
ultimately have ∀ *i* < *length As*. *source* (*Cs*!*i*) = *target* (*Bs*!*i*)
using ∗ *2* **by** (*metis nth-mem nth-zip*)
with ∗ **show** *?case* **by** (*simp add*: *nth-map-conv*)
next
case (*3* α *As* β *Bs*)
then obtain *Cs* **where** ∗:α = β ∧ *length As* = *length Bs* ∧
C = *to-pterm* (*rhs* α) · ⟨*Cs*⟩$_\alpha$ ∧ *length Cs* = *length As* ∧
(∀ *i*<*length As*. *As* ! *i re Bs* ! *i* = *Some* (*Cs* ! *i*))
by (*meson residual-rule-rule*)
then have *length* (*zip As Bs*) = *length As* **by** *simp*
moreover from *3*(*3*) **have** ∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R* **by** *blast*
moreover from *3*(*4*) **have** ∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm S* **by** *blast*
ultimately have *IH*:∀ *i* < *length As*. *source* (*Cs*!*i*) = *target* (*Bs*!*i*)
using ∗ *3* **by** (*metis nth-mem nth-zip*)
from ∗ **have** *source C* = (*rhs* β) · ⟨*map source Cs*⟩$_\beta$
by (*simp add*: *source-apply-subst*)
also have ... = (*rhs* β) · ⟨*map target Bs*⟩$_\beta$
using ∗ *IH* **by** (*metis nth-map-conv*)
finally show *?case* **by** *simp*
next

case (*4-1* α *As v*)
then obtain *Cs* σ **where** $*$:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$ *length Cs* = *length As* $\wedge$
($\forall$ *i*<*length As*. *As* ! *i re* σ (*var-rule* α ! *i*) = *Some* (*Cs* ! *i*))
by (*meson residual-rule-var*)
then obtain *Bs* **where** *Bs*:*length Bs* = *length* (*var-rule* α) $\wedge$
Var v = (*to-pterm* (*lhs* α)) $\cdot$ $\langle Bs \rangle_\alpha$ $\wedge$
($\forall$ *x* $\in$ *set* (*var-rule* α). σ *x* = ($\langle Bs \rangle_\alpha$) *x*)
using *match-lhs-subst* **by** *blast*
from *4-1*(*3*) **have** *as*:$\forall$ *a* $\in$ *set As*. *a* $\in$ *wf-pterm R* **by** *blast*
from *4-1*(*3*) **have** *l*:*length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
with $*$ **have** *well*:$\forall$ *i* < *length As*. σ (*var-rule* α ! *i*) $\in$ *wf-pterm S*
using *4-1*(*4*) **by** (*metis match-well-def vars-to-pterm*)
from *l* **have** *length* (*zip As* (*map* σ (*var-rule* α))) = *length As* **by** *simp*
with *4-1*(*1*,*3*) *well* $*$ *l as* **have** *IH*:$\forall$ *i* < *length As*. *source* (*Cs*!*i*) = *target* (*map* ($\langle Bs \rangle_\alpha$) (*var-rule* α) !*i*)
using *Bs* **by** (*smt length-map nth-map nth-mem nth-zip*)
from $*$ **have** *source C* = (*lhs* α) $\cdot$ $\langle$*map source Cs*$\rangle_\alpha$
by (*simp add*: *source-apply-subst*)
also have ... = (*lhs* α) $\cdot$ $\langle$*map* (*target* $\circ$ ($\langle Bs \rangle_\alpha$)) (*var-rule* α)$\rangle_\alpha$
using $*$ *l IH* **by** (*smt map-eq-conv*′ *map-map*)
also have ... = (*lhs* α) $\cdot$ (*target* $\circ$ ($\langle Bs \rangle_\alpha$))
using *Bs* **by** (*metis* (*no-types*, *lifting*) *apply-lhs-subst-var-rule fun-mk-subst map-map target.simps*(*1*))
also have ... = *target* (*to-pterm* (*lhs* α) $\cdot$ $\langle Bs \rangle_\alpha$)
by (*metis target-empty-apply-subst target-to-pterm to-pterm-empty*)
finally show *?case*
using *Bs* **by** *fastforce*
next
case (*4-2* α *As f Bs*)
then obtain *Cs* σ **where** $*$:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$ *length Cs* = *length As* $\wedge$
($\forall$ *i*<*length As*. *As* ! *i re* σ (*var-rule* α ! *i*) = *Some* (*Cs* ! *i*))
by (*meson residual-rule-fun*)
then obtain *Bs*′ **where** *Bs*′:*length Bs*′ = *length* (*var-rule* α) $\wedge$
Pfun f Bs = (*to-pterm* (*lhs* α)) $\cdot$ $\langle Bs' \rangle_\alpha$ $\wedge$
($\forall$ *x* $\in$ *set* (*var-rule* α). σ *x* = ($\langle Bs' \rangle_\alpha$) *x*)
using *match-lhs-subst* **by** *blast*
from *4-2*(*3*) **have** *as*:$\forall$ *a* $\in$ *set As*. *a* $\in$ *wf-pterm R* **by** *blast*
from *4-2*(*3*) **have** *l*:*length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
with $*$ **have** *well*:$\forall$ *i* < *length As*. σ (*var-rule* α ! *i*) $\in$ *wf-pterm S*
using *4-2*(*4*) **by** (*metis match-well-def vars-to-pterm*)
from *l* **have** *length* (*zip As* (*map* σ (*var-rule* α))) = *length As* **by** *simp*
with *4-2*(*1*,*3*) *well* $*$ *l as* **have** *IH*:$\forall$ *i* < *length As*. *source* (*Cs*!*i*) = *target* (*map* ($\langle Bs' \rangle_\alpha$) (*var-rule* α) !*i*)
using *Bs*′ **by** (*smt length-map nth-map nth-mem nth-zip*)
from $*$ **have** *source C* = (*lhs* α) $\cdot$ $\langle$*map source Cs*$\rangle_\alpha$

```
    by (simp add: source-apply-subst)
  also have ... = (lhs α) · ⟨map (target ∘ (⟨Bs'⟩α)) (var-rule α)⟩α
    using ∗ l IH by (smt map-eq-conv' map-map)
  also have ... = (lhs α) · (target ∘ (⟨Bs'⟩α))
     using Bs' by (metis (no-types, lifting) apply-lhs-subst-var-rule fun-mk-subst
map-map target.simps(1))
  also have ... = target (to-pterm (lhs α) · ⟨Bs'⟩α)
    by (metis target-empty-apply-subst target-to-pterm to-pterm-empty)
  finally show ?case
    using Bs' by fastforce
next
  case (5-1 v α As)
  then obtain Cs σ where ∗:match (Var v) (to-pterm (lhs α)) = Some σ ∧
        C = to-pterm (rhs α) · ⟨Cs⟩α ∧ length Cs = length As ∧
        (∀ i<length As. σ (var-rule α ! i) re As ! i  = Some (Cs ! i))
    by (meson residual-var-rule)
  from 5-1(4) have as:∀ a ∈ set As. a ∈ wf-pterm S by blast
  from 5-1(4) have l:length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  with ∗ have well:∀ i < length As. σ (var-rule α ! i) ∈ wf-pterm R
    using 5-1(3) by (metis match-well-def vars-to-pterm)
  from l have length (zip (map σ (var-rule α)) As) = length As by simp
  with 5-1(1,4) well ∗ l as have IH:∀ i < length As. source (Cs!i) = target (As!i)

    by (smt length-map nth-map nth-mem nth-zip)
  from ∗ have source C = (rhs α) · ⟨map source Cs⟩α
     by (simp add: source-apply-subst)
  also have ... = (rhs α) · ⟨map target As⟩α
    using ∗ IH by (metis (no-types, lifting) map-eq-conv')
  finally show ?case by simp
next
  case (5-2 f Bs α As)
   then obtain Cs σ where ∗:match (Pfun f Bs) (to-pterm (lhs α)) = Some σ ∧
        C = to-pterm (rhs α) · ⟨Cs⟩α ∧ length Cs = length As ∧
        (∀ i<length As. σ (var-rule α ! i) re As ! i  = Some (Cs ! i))
     by (meson residual-fun-rule)
  from 5-2(4) have as:∀ a ∈ set As. a ∈ wf-pterm S by blast
  from 5-2(4) have l:length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  with ∗ have well:∀ i < length As. σ (var-rule α ! i) ∈ wf-pterm R
    using 5-2(3) by (metis match-well-def vars-to-pterm)
  from l have length (zip (map σ (var-rule α)) As) = length As by simp
  with 5-2(1,4) well ∗ l as have IH:∀ i < length As. source (Cs!i) = target (As!i)

    by (smt length-map nth-map nth-mem nth-zip)
  from ∗ have source C = (rhs α) · ⟨map source Cs⟩α
     by (simp add: source-apply-subst)
  also have ... = (rhs α) · ⟨map target As⟩α
    using ∗ IH by (metis (no-types, lifting) map-eq-conv')
```

finally show *?case* **by** *simp*
qed *simp-all*

The following two lemmas are used inside the induction proof for the result $tgt(A\ /\ B) = tgt(B\ /\ A)$. Defining them here, outside the main proof makes them reusable for the symmetric cases of the proof.

lemma *tgt-tgt-rule-var*:
assumes $\bigwedge\sigma\ a\ b\ c\ d.\ match\ (Var\ v)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \sigma \Longrightarrow$
$(a,b) \in set\ (zip\ As\ (map\ \sigma\ (var\text{-}rule\ \alpha))) \Longrightarrow$
$a\ re\ b = Some\ c \Longrightarrow b\ re\ a = Some\ d \Longrightarrow a \in wf\text{-}pterm\ R \Longrightarrow b \in wf\text{-}pterm\ S \Longrightarrow$
$target\ c = target\ d$
$Prule\ \alpha\ As\ re\ Var\ v = Some\ C$
$Var\ v\ re\ Prule\ \alpha\ As = Some\ D$
$Prule\ \alpha\ As \in wf\text{-}pterm\ R$
shows $target\ C = target\ D$
proof−
from *assms(4)* **have** *l*:$length\ As = length\ (var\text{-}rule\ \alpha)$
using *wf-pterm.simps* **by** *fastforce*
from *assms(4)* **have** *as*:$\forall\, a \in set\ As.\ a \in wf\text{-}pterm\ R$ **by** *blast*
from *assms(2,4)* **obtain** σ **where** σ:$match\ (Var\ v)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \sigma$
by (*meson residual-rule-var*)
with *l* **have** *well-def*:$\forall\, i < length\ As.\ \sigma\ (var\text{-}rule\ \alpha\ !\ i) \in wf\text{-}pterm\ S$
using *match-well-def* **by** (*metis vars-to-pterm wf-pterm.intros(1)*)
from *assms(2,4)* σ **obtain** *Cs* **where** *Cs*:
$C = Prule\ \alpha\ Cs \wedge length\ Cs = length\ As$
$(\forall\, i{<}length\ As.\ As\ !\ i\ re\ \sigma\ (var\text{-}rule\ \alpha\ !\ i) = Some\ (Cs\ !\ i))$
by (*metis option.inject residual-rule-var*)
from *assms(3,4)* σ **obtain** *Ds* **where** *Ds*:
$D = to\text{-}pterm\ (rhs\ \alpha) \cdot \langle Ds\rangle_\alpha \wedge length\ Ds = length\ As$
$(\forall\, i{<}length\ As.\ \sigma\ (var\text{-}rule\ \alpha\ !\ i)\ re\ As\ !\ i = Some\ (Ds\ !\ i))$
by (*metis option.inject residual-var-rule*)
from *l* **have** $length\ As = length\ (zip\ As\ (map\ \sigma\ (var\text{-}rule\ \alpha)))$
by *simp*
with *assms(1,4)* σ *l Cs(2) Ds(2) well-def* **have** *IH*:$\forall\, i < length\ As.\ target\ (Cs!i) = target\ (Ds!i)$
using *as* **by** (*smt length-map nth-map nth-mem nth-zip*)
from *Cs* **have** $target\ C = (rhs\ \alpha) \cdot \langle map\ target\ Cs\rangle_\alpha$ **by** *simp*
moreover from *Ds(1)* **have** $target\ D = (rhs\ \alpha) \cdot \langle map\ target\ Ds\rangle_\alpha$
using *target-empty-apply-subst to-pterm-empty* **by** (*metis fun-mk-subst target.simps(1) target-to-pterm*)
ultimately show *?thesis*
using *IH Cs(1) Ds(1)* **by** (*metis nth-map-conv*)
qed

lemma *tgt-tgt-rule-fun*:
assumes $\bigwedge\sigma\ a\ b\ c\ d.\ match\ (Pfun\ f\ Bs)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \sigma \Longrightarrow$
$(a,b) \in set\ (zip\ As\ (map\ \sigma\ (var\text{-}rule\ \alpha))) \Longrightarrow$

a re b = Some c ⟹ b re a = Some d ⟹ a ∈ wf-pterm R ⟹ b ∈ wf-pterm S ⟹
target c = target d
Prule α As re Pfun f Bs = Some C
Pfun f Bs re Prule α As = Some D
Prule α As ∈ wf-pterm R
Pfun f Bs ∈ wf-pterm S
shows *target C = target D*
proof−
from *assms(4)* **have** *l*:*length As = length* (*var-rule α*)
using *wf-pterm.simps* **by** *fastforce*
from *assms(4)* **have** *as*:∀ *a ∈ set As. a ∈ wf-pterm R* **by** *blast*
from *assms(2,4)* **obtain** *σ* **where** *σ*:*match* (*Pfun f Bs*) (*to-pterm* (*lhs α*)) = *Some σ*
by (*meson residual-rule-fun*)
with *assms(2,5) l* **have** *well-def*:∀ *i < length As. σ* (*var-rule α* ! *i*) ∈ *wf-pterm S*
using *match-well-def* **by** (*metis vars-to-pterm*)
from *assms(2,4) σ* **obtain** *Cs* **where** *Cs*:
C = Prule α Cs ∧ *length Cs = length As*
(∀ *i*<*length As. As* ! *i re σ* (*var-rule α* ! *i*) = *Some* (*Cs* ! *i*))
by (*metis option.inject residual-rule-fun*)
from *assms(3,4) σ* **obtain** *Ds* **where** *Ds*:
D = to-pterm (*rhs α*) · ⟨*Ds*⟩$_\alpha$ ∧ *length Ds = length As*
(∀ *i*<*length As. σ* (*var-rule α* ! *i*) *re As* ! *i* = *Some* (*Ds* ! *i*))
by (*metis option.inject residual-fun-rule*)
from *l* **have** *length As = length* (*zip As* (*map σ* (*var-rule α*)))
by *simp*
with *assms(1,4,5) σ l Cs(2) Ds(2) well-def* **have** *IH*:∀ *i < length As. target* (*Cs*!*i*) = *target* (*Ds*!*i*)
using *as* **by** (*smt length-map nth-map nth-mem nth-zip*)
from *Cs* **have** *target C* = (*rhs α*) · ⟨*map target Cs*⟩$_\alpha$ **by** *simp*
moreover from *Ds(1)* **have** *target D* = (*rhs α*) · ⟨*map target Ds*⟩$_\alpha$
using *target-empty-apply-subst to-pterm-empty* **by** (*metis fun-mk-subst target.simps(1) target-to-pterm*)
ultimately show *?thesis*
using *IH Cs(1) Ds(1)* **by** (*metis nth-map-conv*)
qed

lemma *residual-tgt-tgt*:
assumes *A re B = Some C B re A = Some D A ∈ wf-pterm R B ∈ wf-pterm S*
shows *target C = target D*
using *assms* **proof**(*induction A B arbitrary*: *C D rule*:*residual.induct*)
case (*1 x y*)
then show *?case* **by** (*metis option.sel residual.simps(1)*)
next
case (*2 f As g Bs*)
from *2(4)* **have** *as*:∀ *a ∈ set As. a ∈ wf-pterm R* **by** *blast*
from *2(5)* **have** *bs*:∀ *b ∈ set Bs. b ∈ wf-pterm S* **by** *blast*

let *?l* = *length As*
from *2*(*2*) **have** ∗: *f* = *g* ∧ *?l* = *length Bs*
by (*meson residual-fun-fun*)
from *2*(*2*) **obtain** *Cs* **where** *Cs*:
C = *Pfun f Cs* ∧ *length Cs* = *?l* ∧ (∀ *i* < *?l*. *As* ! *i re Bs* ! *i* = *Some* (*Cs* ! *i*))
by (*meson residual-fun-fun*)
from *2*(*3*) **obtain** *Ds* **where** *Ds*:
D = *Pfun g Ds* ∧ *length Ds* = *?l* ∧ (∀ *i* < *?l*. *Bs* ! *i re As* ! *i* = *Some* (*Ds* ! *i*))
using ∗ **by** (*metis residual-fun-fun*)
from ∗ **have** *length* (*zip As Bs*) = *?l* **by** *simp*
with *2*(*1*,*4*,*5*) ∗ *Cs Ds* **have** ∀ *i* < *?l*. *target* (*Cs*!*i*) = *target* (*Ds*!*i*)
using *as bs* **by** (*metis nth-mem nth-zip*)
with ∗ *Cs Ds* **show** *?case*
by (*simp add*: *map-nth-eq-conv*)
next
case (*3* α *As* β *Bs*)
from *3*(*4*) **have** *as*:∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R* **by** *blast*
from *3*(*5*) **have** *bs*:∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm S* **by** *blast*
let *?l* = *length As*
from *3*(*2*,*4*,*5*) **have** ∗: α = β ∧ *?l* = *length Bs*
by (*meson residual-rule-rule*)
from *3*(*2*,*4*,*5*) **obtain** *Cs* **where** *Cs*:
C = *to-pterm* (*rhs* α) · ⟨*Cs*⟩$_\alpha$ ∧ *length Cs* = *?l* ∧ (∀ *i* < *?l*. *As* ! *i re Bs* ! *i* = *Some* (*Cs* ! *i*))
by (*meson residual-rule-rule*)
from *3*(*3*,*4*,*5*) **obtain** *Ds* **where** *Ds*:
D = *to-pterm* (*rhs* α) · ⟨*Ds*⟩$_\alpha$ ∧ *length Ds* = *?l* ∧ (∀ *i* < *?l*. *Bs* ! *i re As* ! *i* = *Some* (*Ds* ! *i*))
using ∗ **by** (*metis residual-rule-rule*)
from ∗ **have** *length* (*zip As Bs*) = *?l* **by** *simp*
with *3*(*1*,*4*,*5*) ∗ *Cs Ds* **have** *IH*:∀ *i* < *?l*. *target* (*Cs*!*i*) = *target* (*Ds*!*i*)
using *as bs* **by** (*metis nth-mem nth-zip*)
from *Cs* **have** *target C* = (*rhs* α) · ⟨*map target Cs*⟩$_\alpha$
using *target-empty-apply-subst to-pterm-empty* **by** (*metis fun-mk-subst target.simps*(*1*) *target-to-pterm*)
moreover from *Ds* **have** *target D* = (*rhs* α) · ⟨*map target Ds*⟩$_\alpha$
using *target-empty-apply-subst to-pterm-empty* **by** (*metis fun-mk-subst target.simps*(*1*) *target-to-pterm*)
ultimately show *?case*
using *IH Cs Ds* **by** (*metis nth-map-conv*)
next
case (*4-1* α *As v*)
then show *?case* **using** *tgt-tgt-rule-var* **by** *fastforce*
next
case (*4-2* α *As f Bs*)
then show *?case* **using** *tgt-tgt-rule-fun* **by** *fastforce*
next
case (*5-1 v* α *As*)
from *5-1*(*1*) **have** *match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ ⟹

```
    (a,b) ∈ set (zip As (map σ (var-rule α))) ⟹
    a re b = Some c ⟹ b re a = Some d ⟹ a ∈ wf-pterm S ⟹ b ∈ wf-pterm
R ⟹
    target c = target d for σ a b c d
    using zip-symm by fastforce
  with 5-1(2,3,5) have target D = target C using tgt-tgt-rule-var by fastforce
  then show ?case by simp
next
  case (5-2 f Bs α As)
  from 5-2(1) have match (Pfun f Bs) (to-pterm (lhs α)) = Some σ ⟹
    (a,b) ∈ set (zip As (map σ (var-rule α))) ⟹
    a re b = Some c ⟹ b re a = Some d ⟹ a ∈ wf-pterm S ⟹ b ∈ wf-pterm
R ⟹
    target c = target d for σ a b c d
    using zip-symm by fastforce
  with 5-2(2,3,4,5) have target D = target C using tgt-tgt-rule-fun by fastforce
  then show ?case by simp
qed simp-all

lemma rule-residual-lhs:
  assumes args:those (map2 (re) As Bs) = Some Cs
    and is-Fun:is-Fun (lhs α) and l:length Bs = length (var-rule α)
  shows Prule α As re (to-pterm (lhs α) · ⟨Bs⟩α) = Some (Prule α Cs)
proof−
  from is-Fun obtain f ts where lhs α = Fun f ts
    by auto
   then have f:to-pterm (lhs α) · ⟨Bs⟩α = Pfun f (map (λt. t · ⟨Bs⟩α) (map
to-pterm ts))
    by simp
  then have match:match ((to-pterm (lhs α)) · ⟨Bs⟩α) (to-pterm (lhs α)) = Some
(⟨Bs⟩α)
    using lhs-subst-trivial by blast
  from l have map (⟨Bs⟩α) (var-rule α) = Bs
    using apply-lhs-subst-var-rule by blast
  with args have those (map2 (re) As (map (⟨Bs⟩α) (var-rule α))) = Some Cs
    by presburger
  then show ?thesis
    using residual.simps(5) match unfolding f by auto
qed

lemma residual-well-defined:
  assumes A ∈ wf-pterm R B ∈ wf-pterm S A re B = Some C
  shows C ∈ wf-pterm R
  using assms proof(induction A B arbitrary:C rule:residual.induct)
  case (1 x y)
  then show ?case
    by (metis option.distinct(1) option.sel residual.simps(1))
next
  case (2 f As g Bs)
```

then obtain *Cs* **where** $f = g \wedge$ *length As* = *length Bs* $\wedge$
C = *Pfun f Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!*i re Bs*!*i* = *Some* (*Cs*!*i*))
by (*meson residual-fun-fun*)
moreover with *2* **have** $i <$ *length As* $\Longrightarrow$ *Cs*!*i* $\in$ *wf-pterm R* **for** *i*
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
ultimately show *?case*
by (*metis in-set-conv-nth wf-pterm.intros*(*2*))
next
case (*3* α *As* β *Bs*)
then obtain *Cs* **where** $\alpha = \beta \wedge$ *length As* = *length Bs* $\wedge$
(*C* = *to-pterm* (*rhs* α) $\cdot$ $\langle$*Cs*$\rangle_\alpha$ $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!*i re Bs*!*i* = *Some* (*Cs*!*i*)))
by (*meson residual-rule-rule*)
moreover with *3* **have** $i <$ *length As* $\Longrightarrow$ *Cs*!*i* $\in$ *wf-pterm R* **for** *i*
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
ultimately show *?case*
by (*metis to-pterm-wf-pterm lhs-subst-well-def*)
next
case (*4-1* α *As v*)
then obtain *Cs* σ **where** $*$:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!*i re* (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*))
by (*meson residual-rule-var*)
from *4-1*(*2*) **have** *wellA*:$\forall\, i <$ *length As*. *As*!*i* $\in$ *wf-pterm R*
by *auto*
from *4-1*(*2*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* $*$ **have** $\forall\, i <$ *length As*. σ (*var-rule* α ! *i*) $\in$ *wf-pterm S*
using *4-1*(*3*) **by** (*metis match-well-def vars-to-pterm*)
with *4-1*(*1*) $*$ *wellA l2* **have** $\forall\, i <$ *length As*. *Cs*!*i* $\in$ *wf-pterm R*
by (*smt* (*z3*) *l length-map nth-map nth-mem nth-zip*)
with *4-1*(*2*) $*$ **show** *?case*
by (*smt* (*verit*) *Inr-not-Inl in-set-conv-nth term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
next
case (*4-2* α *As f Bs*)
then obtain σ *Cs* **where** $*$:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
(*C* = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!*i re* (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*)))
by (*meson residual-rule-fun*)

```
  from 4-2(2) have wellA:∀ i < length As. As!i ∈ wf-pterm R
    by auto
  from 4-2(2) have l: length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  then have l2:length As = length (zip As (map σ (var-rule α)))
    by simp
  from l ∗ have ∀ i < length As. σ (var-rule α ! i) ∈ wf-pterm S
    using 4-2(3) by (metis match-well-def vars-to-pterm)
  with 4-2(1) ∗ wellA l2 have ∀ i < length As. Cs!i ∈ wf-pterm R
    by (smt l length-map nth-map nth-mem nth-zip)
  with 4-2(2) ∗ show ?case
      by (smt (verit, ccfv-threshold) Inr-not-Inl in-set-conv-nth term.distinct(1)
term.inject(2) wf-pterm.cases wf-pterm.intros(3))
next
  case (5-1 v α As)
  then obtain Cs σ where ∗:match (Var v) (to-pterm (lhs α)) = Some σ ∧
        C = to-pterm (rhs α) · ⟨Cs⟩α  ∧
        length Cs = length As ∧
        (∀ i < length As. (σ (var-rule α ! i)) re As!i = Some (Cs!i))
    by (meson residual-var-rule)
  from 5-1(3) have wellA:∀ i < length As. As!i ∈ wf-pterm S
    by auto
  from 5-1(3) have l: length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  then have l2:length As = length (zip (map σ (var-rule α)) As)
    by simp
  from l ∗ have ∀ i < length As. σ (var-rule α ! i) ∈ wf-pterm R
    using 5-1(2) by (metis match-well-def vars-to-pterm)
  with 5-1(1) ∗ wellA l2 have ∀ i < length As. Cs!i ∈ wf-pterm R
    by (smt l length-map nth-map nth-mem nth-zip)
  with ∗ show ?case
    by (metis lhs-subst-well-def to-pterm-wf-pterm)
next
  case (5-2 f Bs α As)
  then obtain Cs σ where ∗:match (Pfun f Bs) (to-pterm (lhs α)) = Some σ ∧
        C = to-pterm (rhs α) · ⟨Cs⟩α  ∧
        length Cs = length As ∧
        (∀ i < length As. (σ (var-rule α ! i)) re As!i = Some (Cs!i))
    by (meson residual-fun-rule)
  from 5-2(3) have wellA:∀ i < length As. As!i ∈ wf-pterm S
    by auto
  from 5-2(3) have l: length As = length (var-rule α)
    using wf-pterm.simps by fastforce
  then have l2:length As = length (zip (map σ (var-rule α)) As)
    by simp
  from l ∗ have ∀ i < length As. σ (var-rule α ! i) ∈ wf-pterm R
    using 5-2(2) by (metis match-well-def vars-to-pterm)
  with 5-2(1) ∗ wellA l2 have ∀ i < length As. Cs!i ∈ wf-pterm R
    by (smt l length-map nth-map nth-mem nth-zip)
```

with ∗ **show** *?case*
by (*metis lhs-subst-well-def to-pterm-wf-pterm*)
qed *simp-all*

no-notation *sup* (**infixl** ⊔ *65*)

3.2 Join

fun *join* :: ('*f*, '*v*) *pterm* ⇒ ('*f*,'*v*) *pterm* ⇒ ('*f*,'*v*) *pterm option* (**infixr** ⊔ *70*)
where
Var x ⊔ *Var y* =
(*if x* = *y then Some* (*Var x*) *else None*)
| *Pfun f As* ⊔ *Pfun g Bs* =
(*if* (*f* = *g* ∧ *length As* = *length Bs*) *then*
(*case those* (*map2* (⊔) *As Bs*) *of*
Some xs ⇒ *Some* (*Pfun f xs*)
| *None* ⇒ *None*)
else None)
| *Prule* α *As* ⊔ *Prule* β *Bs* =
(*if* α = β *then*
(*case those* (*map2* (⊔) *As Bs*) *of*
Some xs ⇒ *Some* (*Prule* α *xs*)
| *None* ⇒ *None*)
else None)
| *Prule* α *As* ⊔ *B* =
(*case match B* (*to-pterm* (*lhs* α)) *of*
None ⇒ *None*
| *Some* σ ⇒
(*case those* (*map2* (⊔) *As* (*map* σ (*var-rule* α))) *of*
Some xs ⇒ *Some* (*Prule* α *xs*)
| *None* ⇒ *None*))
| *A* ⊔ *Prule* α *Bs* =
(*case match A* (*to-pterm* (*lhs* α)) *of*
None ⇒ *None*
| *Some* σ ⇒
(*case those* (*map2* (⊔) (*map* σ (*var-rule* α)) *Bs*) *of*
Some xs ⇒ *Some* (*Prule* α *xs*)
| *None* ⇒ *None*))
| *A* ⊔ *B* = *None*

lemma *join-sym*: *A* ⊔ *B* = *B* ⊔ *A*
proof(*induct rule*:*join.induct*)
case (*2 f As g Bs*)
then show *?case* **proof**(*cases f* = *g* ∧ *length As* = *length Bs*)
case *True*
with *2* **have** ∀(*a*,*b*) ∈ *set* (*zip As Bs*). *a* ⊔ *b* = *b* ⊔ *a*
by *auto*
with *True* **have** (*map2* (⊔) *As Bs*) = (*map2* (⊔) *Bs As*)
by (*smt case-prod-unfold map-eq-conv' map-fst-zip map-snd-zip nth-mem*)

then show *?thesis* **using** *2* **unfolding** *join.simps*
by *auto*
qed *auto*
next
case (*3* α *As* β *Bs*)
then show *?case* **proof**(*cases* $\alpha = \beta$)
case *True*
with *3* **have** $*{:}\forall (a,b) \in$ *set* (*zip As Bs*). $a \sqcup b = b \sqcup a$
by *auto*
have *length* (*map2* ($\sqcup$) *As Bs*) = *length* (*map2* ($\sqcup$) *Bs As*)
by *auto*
with $*$ **have** (*map2* ($\sqcup$) *As Bs*) = (*map2* ($\sqcup$) *Bs As*)
by (*smt fst-conv length-map length-zip map-eq-conv' min-less-iff-conj nth-mem nth-zip prod.case-eq-if snd-conv*)
then show *?thesis* **using** *3* **unfolding** *join.simps*
by *auto*
qed *auto*
next
case (*4-1* α *As v*)
then show *?case* **proof**(*cases match* (*Var v*) (*to-pterm* (*lhs* α)) = *None*)
case *False*
then obtain σ **where** *sigma*:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ
by *blast*
with *4-1* **have** $*{:}\forall (a,b) \in$ *set* (*zip As* (*map* σ (*var-rule* α))). $a \sqcup b = b \sqcup a$
by *auto*
have *length* (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = *length* (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *As*)
by *auto*
with $*$ **have** (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *As*)
by (*smt fst-conv length-map length-zip map-eq-conv' min-less-iff-conj nth-mem nth-zip prod.case-eq-if snd-conv*)
then show *?thesis* **unfolding** *join.simps sigma*
by *simp*
qed *simp*
next
case (*4-2* α *As f Bs*)
then show *?case* **proof**(*cases match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *None*)
case *False*
then obtain σ **where** *sigma*:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ
by *blast*
with *4-2* **have** $*{:}\forall (a,b) \in$ *set* (*zip As* (*map* σ (*var-rule* α))). $a \sqcup b = b \sqcup a$
by *auto*
have *length* (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = *length* (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *As*)
by *auto*
with $*$ **have** (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *As*)
by (*smt fst-conv length-map length-zip map-eq-conv' min-less-iff-conj nth-mem*

nth-zip prod.case-eq-if snd-conv)
  then show *?thesis* **unfolding** *join.simps sigma*
    by *simp*
 qed *simp*
next
 case (*5-1 v α Bs*)
 then show *?case* **proof**(*cases match* (*Var v*) (*to-pterm* (*lhs α*)) = *None*)
  case *False*
  then obtain *σ* **where** *sigma*:*match* (*Var v*) (*to-pterm* (*lhs α*)) = *Some σ*
    by *blast*
  with *5-1* **have** ∗:∀ (*a*,*b*) ∈ *set* (*zip* (*map σ* (*var-rule α*)) *Bs*). *a* ⊔ *b* = *b* ⊔ *a*
    by *auto*
  have *length* (*map2* (⊔) (*map σ* (*var-rule α*)) *Bs*) = *length* (*map2* (⊔) *Bs* (*map σ* (*var-rule α*)))
    by *auto*
  with ∗ **have** (*map2* (⊔) (*map σ* (*var-rule α*)) *Bs*) = (*map2* (⊔) *Bs* (*map σ* (*var-rule α*)))
    by (*smt fst-conv length-map length-zip map-eq-conv*′ *min-less-iff-conj nth-mem nth-zip prod.case-eq-if snd-conv*)
  then show *?thesis* **unfolding** *join.simps sigma*
    by *simp*
 qed *simp*
next
 case (*5-2 f As α Bs*)
 then show *?case* **proof**(*cases match* (*Pfun f As*) (*to-pterm* (*lhs α*)) = *None*)
  case *False*
  then obtain *σ* **where** *sigma*:*match* (*Pfun f As*) (*to-pterm* (*lhs α*)) = *Some σ*
    by *blast*
  with *5-2* **have** ∗:∀ (*a*,*b*) ∈ *set* (*zip* (*map σ* (*var-rule α*)) *Bs*). *a* ⊔ *b* = *b* ⊔ *a*
    by *auto*
  have *length* (*map2* (⊔) (*map σ* (*var-rule α*)) *Bs*) = *length* (*map2* (⊔) *Bs* (*map σ* (*var-rule α*)))
    by *auto*
  with ∗ **have** (*map2* (⊔) (*map σ* (*var-rule α*)) *Bs*) = (*map2* (⊔) *Bs* (*map σ* (*var-rule α*)))
    by (*smt fst-conv length-map length-zip map-eq-conv*′ *min-less-iff-conj nth-mem nth-zip prod.case-eq-if snd-conv*)
  then show *?thesis* **unfolding** *join.simps sigma*
    by *simp*
 qed *simp*
qed *simp-all*

lemma *join-with-source*:
 assumes *A* ∈ *wf-pterm R*
 shows *A* ⊔ *to-pterm* (*source A*) = *Some A*
using *assms* **proof**(*induct A*)
 case (*2 As f*)
 then have ∀ *i* < *length As*. (*map2* (⊔) *As* (*map to-pterm* (*map source As*)))!*i* = *Some* (*As*!*i*)

by *simp*
then have *those* (*map2* (⊔) *As* (*map to-pterm* (*map source As*))) = *Some As*
by (*simp add*: *those-some*)
then show *?case*
by *simp*
next
case (*3* α *As*)
then have ∀ *i* < *length As*. (*map2* (⊔) *As* (*map to-pterm* (*map source As*)))!*i* = *Some* (*As*!*i*)
by *simp*
then have *IH*:*those* (*map2* (⊔) *As* (*map to-pterm* (*map source As*))) = *Some As*
by (*simp add*: *those-some*)
from *3*(*1*) **have** *match*:*match* (*to-pterm* (*source* (*Prule* α *As*))) (*to-pterm* (*lhs* α)) = *Some* (⟨*map* (*to-pterm* ∘ *source*) *As*⟩$_\alpha$)
by (*metis* (*no-types*, *lifting*) *fun-mk-subst lhs-subst-trivial map-map source.simps*(*3*) *to-pterm.simps*(*1*) *to-pterm-subst*)
from *3*(*2*) **have** (*map* (⟨*map* (*to-pterm* ∘ *source*) *As*⟩$_\alpha$) (*var-rule* α)) = *map* (*to-pterm* ∘ *source*) *As*
by (*metis apply-lhs-subst-var-rule length-map*)
with *IH match join.simps*(*4*,*5*) **show** *?case* **by**(*cases source* (*Prule* α *As*)) *simp-all*
qed *simp*

context *no-var-lhs*
begin

lemma *join-subst*:
assumes *B* ∈ *wf-pterm R* **and** ∀ *x* ∈ *vars-term B*. ϱ *x* ∈ *wf-pterm R*
and ∀ *x* ∈ *vars-term B*. *source* (ϱ *x*) = σ *x*
shows (*B* · (*to-pterm* ∘ σ)) ⊔ ((*to-pterm* (*source B*)) · ϱ) = *Some* (*B* · ϱ)
using *assms* **proof**(*induct B*)
case (*1 x*)
then show *?case* **unfolding** *eval-term.simps source.simps to-pterm.simps o-apply*
using *join-with-source* **by** (*metis term.set-intros*(*3*) *join-sym*)
next
case (*2 ts f*)
{fix *i* **assume** *i*:*i* < *length ts*
with *2* **have** ((*ts*!*i*) · (*to-pterm* ∘ σ)) ⊔ ((*to-pterm* (*source* (*ts*!*i*))) · ϱ) = *Some* (*ts*!*i* · ϱ)
by (*meson nth-mem term.set-intros*(*4*))
then have *map2* (⊔) (*map* (λ*t*. *t* · (*to-pterm* ∘ σ)) *ts*) (*map* (λ*t*. *t* · ϱ) (*map to-pterm* (*map source ts*)))!*i* = *Some* ((*map* (λ*t*. *t* · ϱ) *ts*)!*i*)
using *i* **by** *fastforce*
}
then have *those* (*map2* (⊔) (*map* (λ*t*. *t* · (*to-pterm* ∘ σ)) *ts*) (*map* (λ*t*. *t* · ϱ) (*map to-pterm* (*map source ts*)))) = *Some* (*map* (λ*t*. *t* · ϱ) *ts*)
using *those-some* **by** (*smt* (*verit*) *length-map length-zip min.idem*)
then show *?case*
unfolding *source.simps to-pterm.simps eval-term.simps* **using** *join.simps*(*2*)

by *auto*
next
case (*3* α *As*)
from *3*(*1*) *no-var-lhs* **obtain** *f ts* **where** *f*:*lhs* α = *Fun f ts*
by *fastforce*
obtain τ **where** *match*:*match* (*to-pterm* (*lhs* α · $\langle$*map source As*$\rangle_\alpha$) · ϱ) (*to-pterm* (*lhs* α)) = *Some* τ
and τ:($\forall$ *x*$\in$*vars-term* (*lhs* α). τ *x* = ((*to-pterm* ∘ $\langle$*map source As*$\rangle_\alpha$) ∘$_s$ ϱ) *x*)
using *match-complete*′ **unfolding** *to-pterm-subst* **by** (*smt* (*verit*, *best*) *set-vars-term-list subst-subst vars-to-pterm*)
{fix *i* **assume** *i*:*i* < *length* (*var-rule* α)
let *?x* =*var-rule* α ! *i*
have ((*to-pterm* ∘ $\langle$*map source As*$\rangle_\alpha$) ∘$_s$ ϱ) *?x* = *to-pterm* (*source* (*As*!*i*)) · ϱ
using *i 3*(*2*) **by** (*simp add*: *mk-subst-distinct subst-compose-def*)
moreover from *3* **have** ((*As*!*i*) · (*to-pterm* ∘ σ)) ⊔ (*to-pterm* (*source* (*As*!*i*)) · ϱ) = *Some* ((*As*!*i*) · ϱ)
by (*metis* (*mono-tags*, *lifting*) *i nth-mem term.set-intros*(*4*))
ultimately have ((*As*!*i*) · (*to-pterm* ∘ σ)) ⊔ (τ *?x*) = *Some* ((*As*!*i*) · ϱ)
using τ **by** (*metis comp-apply i nth-mem set-remdups set-rev set-vars-term-list*)
then have (*map2* (⊔) (*map* (λ*t*. *t* · (*to-pterm* ∘ σ)) *As*) (*map* τ (*var-rule* α)))!*i* = *Some* ((*map* (λ*t*. *t* · ϱ) *As*)!*i*)
using *3*(*2*) *i* **by** *auto*
}
then have *those* (*map2* (⊔) (*map* (λ*t*. *t* · (*to-pterm* ∘ σ)) *As*) (*map* τ (*var-rule* α))) = *Some* (*map* (λ*t*. *t* · ϱ) *As*)
by (*simp add*: *3*(*2*) *those-some*)
then show *?case*
using *match* **unfolding** *source.simps to-pterm.simps eval-term.simps f* **using** *join.simps*(*5*) *f* **by** *auto*
qed

end

lemma *join-same*:
shows *A* ⊔ *A* = *Some A*
proof(*induct A*)
case (*Pfun f As*)
{fix *i* **assume** *i*:*i* < *length As*
with *Pfun* **have** *As*!*i* ⊔ *As*!*i* = *Some* (*As*!*i*) **by** *simp*
with *i* **have** (*map2* (⊔) *As As*)!*i* = *Some* (*As*!*i*) **by** *simp*
}
then have *those* (*map2* (⊔) *As As*) = *Some As*
by (*simp add*: *those-some*)
then show *?case* **unfolding** *join.simps* **by** *simp*
next
case (*Prule* α *As*)
{fix *i* **assume** *i*:*i* < *length As*
with *Prule* **have** *As*!*i* ⊔ *As*!*i* = *Some* (*As*!*i*) **by** *simp*
with *i* **have** (*map2* (⊔) *As As*)!*i* = *Some* (*As*!*i*) **by** *simp*

```
  }
  then have those (map2 (⊔) As As) = Some As
    by (simp add: those-some)
  then show ?case unfolding join.simps by simp
qed simp
```

Analogous to residuals there are 6 lemmas corresponding to the step cases in induction proofs for joins.

```
lemma join-fun-fun:
  assumes (Pfun f As) ⊔ (Pfun g Bs) = Some C
  shows f = g ∧ length As = length Bs ∧
        (∃ Cs. C = Pfun f Cs ∧
        length Cs = length As ∧
        (∀ i < length As. As!i ⊔ Bs!i = Some (Cs!i)))
proof−
  have ∗:f = g ∧ length As = length Bs
    using assms join.simps(2) by (metis option.simps(3))
  then obtain Cs where Cs:those (map2 (⊔) As Bs) = Some Cs
    using assms option.simps(3) option.simps(4) by fastforce
  hence ∀ i < length As. As!i ⊔ Bs!i = Some (Cs!i)
    using ∗ those-some2 by fastforce
  with ∗ Cs assms(1) show ?thesis
    using length-those by fastforce
qed

lemma join-rule-rule:
  assumes (Prule α As) ⊔ (Prule β Bs) = Some C
          (Prule α As) ∈ wf-pterm R
          (Prule β Bs) ∈ wf-pterm R
  shows α = β ∧ length As = length Bs ∧
        (∃ Cs. C = Prule α Cs ∧
        length Cs = length As ∧
        (∀ i < length As. As!i ⊔ Bs!i = Some (Cs!i)))
proof−
  have α = β
    using assms(1) join.simps(3) by (metis option.simps(3))
  with assms(2,3) have l: length As = length Bs
    using length-args-well-Prule by blast
  from ‹α = β› obtain Cs where Cs:those (map2 (⊔) As Bs) = Some Cs
    using assms by fastforce
  hence ∀ i < length As. As!i ⊔ Bs!i = Some (Cs!i)
    using l those-some2 by fastforce
  with ‹α = β› l Cs assms(1) show ?thesis
    using length-those by fastforce
qed

lemma join-rule-var:
  assumes (Prule α As) ⊔ (Var x) = Some C
          (Prule α As) ∈ wf-pterm R
```

shows $\exists\sigma.\ match\ (Var\ x)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \sigma \wedge$
$(\exists\, Cs.\ C = Prule\ \alpha\ Cs \wedge$
$length\ Cs = length\ As \wedge$
$(\forall\, i < length\ As.\ As!i \sqcup (\sigma\ (var\text{-}rule\ \alpha\ !\ i)) = Some\ (Cs!i)))$
proof−
from *assms*(*2*) **have** *l*:*length As* = *length* (*var-rule* α)
using *wf-pterm.cases* **by** *auto*
obtain σ **where** σ:*match* (*Var x*) (*to-pterm* (*lhs* α)) = *Some* σ
using *assms*(*1*) **by** *fastforce*
then obtain *Cs* **where** *Cs*:*those* (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = *Some Cs*
using *assms*(*1*) **by** *fastforce*
with *l* **have** *l2*:*length Cs* = *length As*
using *length-those* **by** *fastforce*
from *Cs* **have** $\forall\, i <$ *length As*. *As*!*i* $\sqcup$ (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*)
using *l those-some2* **by** *fastforce*
with σ *Cs assms*(*1*) *l2* **show** *?thesis* **by** *simp*
qed

lemma *join-rule-fun*:
assumes (*Prule* α *As*) $\sqcup$ (*Pfun f Bs*) = *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
shows $\exists\sigma.\ match\ (Pfun\ f\ Bs)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \sigma \wedge$
$(\exists\, Cs.\ C = Prule\ \alpha\ Cs \wedge$
$length\ Cs = length\ As \wedge$
$(\forall\, i < length\ As.\ As!i \sqcup (\sigma\ (var\text{-}rule\ \alpha\ !\ i)) = Some\ (Cs!i)))$
proof−
from *assms*(*2*) **have** *l*:*length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
obtain σ **where** σ:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ
using *assms*(*1*) **by** *fastforce*
then obtain *Cs* **where** *Cs*:*those* (*map2* ($\sqcup$) *As* (*map* σ (*var-rule* α))) = *Some Cs*
using *assms*(*1*) **by** *fastforce*
with *l* **have** *l2*:*length Cs* = *length As*
using *length-those* **by** *fastforce*
from *Cs* **have** $\forall\, i <$ *length As*. *As*!*i* $\sqcup$ (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*)
using *l those-some2* **by** *fastforce*
with σ *Cs assms*(*1*) *l2* **show** *?thesis* **by** *auto*
qed

lemma *join-wf-pterm*:
assumes *A* $\sqcup$ *B* = *Some C*
and *A* $\in$ *wf-pterm R* **and** *B* $\in$ *wf-pterm R*
shows *C* $\in$ *wf-pterm R*
using *assms* **proof**(*induct A B arbitrary*:*C rule*:*join.induct*)
case (*1 x y*)
then show *?case*
by (*metis join.simps*(*1*) *option.distinct*(*1*) *option.sel*)

next
case (*2 f As g Bs*)
then obtain *Cs* **where** $f = g \wedge$ *length As* = *length Bs* $\wedge$
C = *Pfun f Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!$i \sqcup$ *Bs*!i = *Some* (*Cs*!i))
by (*meson join-fun-fun*)
moreover with *2* **have** $i <$ *length As* $\Longrightarrow$ *Cs*!$i \in$ *wf-pterm R* **for** i
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
ultimately show *?case*
by (*metis in-set-conv-nth wf-pterm.intros*(*2*))
next
case (*3* α *As* β *Bs*)
then obtain *Cs* **where** $\alpha = \beta \wedge$ *length As* = *length Bs* $\wedge$
(*C* = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!$i \sqcup$ *Bs*!i = *Some* (*Cs*!i)))
by (*meson join-rule-rule*)
moreover with *3* **have** $i <$ *length As* $\Longrightarrow$ *Cs*!$i \in$ *wf-pterm R* **for** i
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
moreover from *3*(*3*) **have** *to-rule* $\alpha \in R$
using *wf-pterm.simps* **by** *fastforce*
ultimately show *?case*
by (*smt* (*verit*, *best*) *3.prems*(*2*) *in-set-idx old.sum.distinct*(*2*) *term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
next
case (*4-1* α *As v*)
then obtain *Cs* σ **where** $*$:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*!$i \sqcup$ (σ (*var-rule* α ! i)) = *Some* (*Cs*!i))
by (*meson join-rule-var*)
from *4-1*(*3*) **have** *wellA*:$\forall\, i <$ *length As*. *As*!$i \in$ *wf-pterm R*
by *auto*
from *4-1*(*3*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* $*$ **have** $\forall\, i <$ *length As*. σ (*var-rule* α ! i) $\in$ *wf-pterm R*
using *4-1*(*4*) **by** (*metis match-well-def vars-to-pterm*)
with *4-1*(*1*) $*$ *wellA l2* **have** $\forall\, i <$ *length As*. *Cs*!$i \in$ *wf-pterm R*
by (*smt* (*z3*) *l length-map nth-map nth-mem nth-zip*)
with *4-1*(*3*) $*$ **show** *?case*
by (*smt* (*verit*) *Inr-not-Inl in-set-conv-nth term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
next
case (*4-2* α *As f Bs*)

then obtain σ *Cs* **where** $*$:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
(C = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
($\forall\, i <$ *length As*. *As*!i $\sqcup$ (σ (*var-rule* α ! i)) = *Some* (*Cs*!i)))
by (*meson join-rule-fun*)
from *4-2*(*3*) **have** *wellA*:$\forall\, i <$ *length As*. *As*!$i \in$ *wf-pterm R*
by *auto*
from *4-2*(*3*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* $*$ **have** $\forall\, i <$ *length As*. σ (*var-rule* α ! i) $\in$ *wf-pterm R*
using *4-2*(*4*) **by** (*metis match-well-def vars-to-pterm*)
with *4-2*(*1*) $*$ *wellA l2* **have** $\forall\, i <$ *length As*. *Cs*!$i \in$ *wf-pterm R*
by (*smt l length-map nth-map nth-mem nth-zip*)
with *4-2*(*3*) $*$ **show** *?case*
by (*smt* (*verit*, *ccfv-threshold*) *Inr-not-Inl in-set-conv-nth term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
next
case (*5-1 v* α *As*)
then obtain *Cs* σ **where** $*$:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
($\forall\, i <$ *length As*. (σ (*var-rule* α ! i)) $\sqcup$ *As*!i = *Some* (*Cs*!i))
using *join-rule-var* **by** (*metis join-sym*)
from *5-1*(*4*) **have** *wellA*:$\forall\, i <$ *length As*. *As*!$i \in$ *wf-pterm R*
by *auto*
from *5-1*(*4*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* $*$ **have** $\forall\, i <$ *length As*. σ (*var-rule* α ! i) $\in$ *wf-pterm R*
using *5-1*(*3*) **by** (*metis match-well-def vars-to-pterm*)
with *5-1*(*1*) $*$ *wellA l2 l* **have** $\forall\, i <$ *length As*. *Cs*!$i \in$ *wf-pterm R*
by (*smt* (*verit*, *del-insts*) *length-map nth-map nth-mem nth-zip zip-symm*)
with *5-1*(*4*) $*$ **show** *?case*
by (*smt* (*verit*) *Inr-not-Inl in-set-conv-nth term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
next
case (*5-2 f Bs* α *As*)
then obtain *Cs* σ **where** $*$:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ $\wedge$
C = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
($\forall\, i <$ *length As*. (σ (*var-rule* α ! i)) $\sqcup$ *As*!i = *Some* (*Cs*!i))
using *join-sym join-rule-fun* **by** *metis*
from *5-2*(*4*) **have** *wellA*:$\forall\, i <$ *length As*. *As*!$i \in$ *wf-pterm R*
by *auto*
from *5-2*(*4*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*

then have *l2*:*length As* = *length* (*zip* (*map* σ (*var-rule* α)) *As*)
by *simp*
from *l* ∗ **have** ∀ *i* < *length As*. σ (*var-rule* α ! *i*) ∈ *wf-pterm R*
using *5-2*(*3*) **by** (*metis match-well-def vars-to-pterm*)
with *5-2*(*1*) ∗ *wellA l2* **have** ∀ *i* < *length As*. *Cs*!*i* ∈ *wf-pterm R*
by (*smt l length-map nth-map nth-mem nth-zip*)
with ∗ **show** *?case*
by (*metis 5-2.prems*(*3*) *Inl-inject Inr-not-Inl in-set-idx l term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.cases wf-pterm.intros*(*3*))
qed *auto*

lemma *source-join*:
assumes *A* ⊔ *B* = *Some C*
and *A* ∈ *wf-pterm R* **and** *B* ∈ *wf-pterm R*
shows *co-initial A C*
using *assms* **proof**(*induct A B arbitrary*:*C rule*:*join.induct*)
case (*1 x y*)
then show *?case*
by (*metis join.simps*(*1*) *option.discI option.sel*)
next
case (*2 f As g Bs*)
then obtain *Cs* **where** *f* = *g* ∧ *length As* = *length Bs* ∧
C = *Pfun f Cs* ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. *As*!*i* ⊔ *Bs*!*i* = *Some* (*Cs*!*i*))
by (*meson join-fun-fun*)
moreover with *2* **have** *i* < *length As* ⟹ *co-initial* (*As*!*i*) (*Cs*!*i*) **for** *i*
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
ultimately show *?case*
by (*simp add*: *nth-map-conv*)
next
case (*3* α *As* β *Bs*)
then obtain *Cs* **where** α = β ∧ *length As* = *length Bs* ∧
(*C* = *Prule* α *Cs* ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. *As*!*i* ⊔ *Bs*!*i* = *Some* (*Cs*!*i*)))
by (*meson join-rule-rule*)
moreover with *3* **have** *i* < *length As* ⟹ *co-initial* (*As*!*i*) (*Cs*!*i*) **for** *i*
using *fun-well-arg* **by** (*metis* (*no-types*, *opaque-lifting*) *fst-conv in-set-zip nth-mem snd-conv*)
ultimately show *?case*
by (*metis* (*mono-tags*, *lifting*) *map-eq-conv*′ *source.simps*(*3*))
next
case (*4-1* α *As v*)
then obtain *Cs* σ **where** ∗:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ ∧
C = *Prule* α *Cs* ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. *As*!*i* ⊔ (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*))

by (*meson join-rule-var*)
from *4-1*(*3*) **have** *wellA*:∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
by *auto*
from *4-1*(*3*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* ∗ **have** ∀ *i* < *length As*. σ (*var-rule* α ! *i*) ∈ *wf-pterm R*
using *4-1*(*4*) **by** (*metis match-well-def vars-to-pterm*)
with *4-1*(*1*) ∗ *wellA l2* **have** ∀ *i* < *length As*. *co-initial* (*As*!*i*) (*Cs*!*i*)
by (*smt* (*z3*) *l length-map nth-map nth-mem nth-zip*)
with *4-1*(*3*) ∗ **show** *?case*
by (*metis nth-map-conv source.simps*(*3*))
next
case (*4-2* α *As f Bs*)
then obtain σ *Cs* **where** ∗:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ ∧
(*C* = *Prule* α *Cs* ∧
length Cs = *length As* ∧
(∀ *i* < *length As*. *As*!*i* ⊔ (σ (*var-rule* α ! *i*)) = *Some* (*Cs*!*i*)))
by (*meson join-rule-fun*)
from *4-2*(*3*) **have** *wellA*:∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
by *auto*
from *4-2*(*3*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* ∗ **have** ∀ *i* < *length As*. σ (*var-rule* α ! *i*) ∈ *wf-pterm R*
using *4-2*(*4*) **by** (*metis match-well-def vars-to-pterm*)
with *4-2*(*1*) ∗ *wellA l2* **have** ∀ *i* < *length As*. *co-initial* (*As*!*i*) (*Cs*!*i*)
by (*smt l length-map nth-map nth-mem nth-zip*)
with *4-2*(*3*) ∗ **show** *?case*
by (*metis nth-map-conv source.simps*(*3*))
next
case (*5-1 v* α *As*)
then obtain *Cs* σ **where** ∗:*match* (*Var v*) (*to-pterm* (*lhs* α)) = *Some* σ
C = *Prule* α *Cs*
length Cs = *length As* ∧
(∀ *i* < *length As*. (σ (*var-rule* α ! *i*)) ⊔ *As*!*i* = *Some* (*Cs*!*i*))
using *join-rule-var* **by** (*metis join-sym*)
from *5-1*(*4*) **have** *wellA*:∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
by *auto*
from *5-1*(*4*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* ∗ **have** ∀ *i* < *length As*. σ (*var-rule* α ! *i*) ∈ *wf-pterm R*
using *5-1*(*3*) **by** (*metis match-well-def vars-to-pterm*)
with *5-1*(*1*) ∗ *wellA l2 l* **have** *IH*:∀ *i* < *length As*. *co-initial* ((*map* σ (*var-rule* α))!*i*) (*Cs*!*i*)

by (*smt* (*verit*, *del-insts*) *length-map nth-map nth-mem nth-zip zip-symm*)
moreover have *v*:*Var v* = (*to-pterm* (*lhs* α)) · ⟨(*map* σ (*var-rule* α))⟩$_\alpha$
using ∗ **by** (*smt* (*verit*, *ccfv-threshold*) *apply-lhs-subst-var-rule map-eq-conv match-lhs-subst*)
show *?case* **using** *IH l* **unfolding** *v* ∗(*2*) *source.simps*
by (*metis* ∗(*1*,*3*) *fun-mk-subst length-map lhs-subst-trivial nth-map-conv option.inject source.simps*(*1*) *source-apply-subst source-to-pterm to-pterm-wf-pterm v*)
next
case (*5-2 f Bs* α *As*)
then obtain *Cs* σ **where** ∗:*match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) = *Some* σ
C = *Prule* α *Cs*
length Cs = *length As* ∧
(∀ *i* < *length As*. (σ (*var-rule* α ! *i*)) ⊔ *As*!*i* = *Some* (*Cs*!*i*))
using *join-rule-fun* **by** (*metis join-sym*)
from *5-2*(*4*) **have** *wellA*:∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
by *auto*
from *5-2*(*4*) **have** *l*: *length As* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
then have *l2*:*length As* = *length* (*zip As* (*map* σ (*var-rule* α)))
by *simp*
from *l* ∗ **have** ∀ *i* < *length As*. σ (*var-rule* α ! *i*) ∈ *wf-pterm R*
using *5-2*(*3*) **by** (*metis match-well-def vars-to-pterm*)
with *5-2*(*1*) ∗ *wellA l2 l* **have** *IH*:∀ *i* < *length As*. *co-initial* ((*map* σ (*var-rule* α))!*i*) (*Cs*!*i*)
by (*smt* (*verit*, *del-insts*) *length-map nth-map nth-mem nth-zip zip-symm*)
moreover have *f*:*Pfun f Bs* = (*to-pterm* (*lhs* α)) · ⟨(*map* σ (*var-rule* α))⟩$_\alpha$
using ∗ **by** (*smt* (*verit*, *ccfv-threshold*) *apply-lhs-subst-var-rule map-eq-conv match-lhs-subst*)
show *?case* **using** *IH l* **unfolding** *f* ∗(*2*) *source.simps*
by (*metis* ∗(*3*) *fun-mk-subst length-map nth-map-conv source.simps*(*1*) *source-apply-subst source-to-pterm to-pterm-wf-pterm*)
qed *auto*

lemma *join-pterm-subst-Some*:
fixes *A B*::('*f*, '*v*) *pterm*
assumes (*A* · σ) ⊔ (*A* · τ) = *Some B*
shows ∃ ϱ. (∀ *x* ∈ *vars-term A*. σ *x* ⊔ τ *x* = *Some* (ϱ *x*)) ∧ *B* = *A* · ϱ ∧ *match B A* = *Some* ϱ
proof−
let *?join-var*=λ*x*. *the* (σ *x* ⊔ τ *x*)
let *?ϱ*=*mk-subst Var* (*zip* (*vars-distinct A*) (*map ?join-var* (*vars-distinct A*)))
from *assms* **have** *B* = *A* · *?ϱ* ∧ (∀ *x* ∈ *vars-term A*. σ *x* ⊔ τ *x* = *Some* (*?ϱ x*)) ∧ *match B A* = *Some ?ϱ* **proof**(*induct A arbitrary*: *B*)
case (*Var x*)
then show *?case*
by (*smt* (*verit*) *comp-apply eval-term.simps*(*1*) *in-set-conv-nth in-set-simps*(*2*) *length-map map-nth-conv match-trivial mem-Collect-eq mk-subst-not-mem mk-subst-same option.sel remdups-id-iff-distinct set-vars-term-list single-var*

singleton-rev-conv subsetI subst-domain-def vars-term-list.simps(1))
next
case (*Pfun f As*)
let *?ϱ=mk-subst Var* (*zip* (*vars-distinct* (*Pfun f As*)) (*map ?join-var* (*vars-distinct* (*Pfun f As*))))
have *rho-domain:subst-domain ?ϱ* ⊆ *vars-term* (*Pfun f As*)
by (*smt* (*verit*, *del-insts*) *comp-apply mem-Collect-eq mk-subst-not-mem set-remdups set-rev set-vars-term-list subsetI subst-domain-def*)
from *Pfun(2)* **have** *those* (*map2* (⊔) (*map* (λ*a*. *a* · *σ*) *As*) (*map* (λ*a*. *a* · *τ*) *As*)) ≠ *None*
unfolding *eval-term.simps join.simps* **using** *option.simps(4)* **by** *fastforce*
then obtain *Bs* **where** *Bs:those* (*map2* (⊔) (*map* (λ*a*. *a* · *σ*) *As*) (*map* (λ*a*. *a* · *τ*) *As*)) = *Some Bs length As* = *length Bs*
using *length-those* **by** *fastforce*
{fix *i* **assume** *i* < *length As*
with *Bs* **have** *Bs-i:*((*As*!*i*) · *σ*) ⊔ ((*As*!*i*) · *τ*) = *Some* (*Bs*!*i*)
using *those-some2* **by** *fastforce*
}note *Bs-i=this*
{fix *i* **assume** *i:i* < *length As*
let *?ϱi=mk-subst Var* (*zip* (*vars-distinct* (*As*!*i*)) (*map ?join-var* (*vars-distinct* (*As*!*i*))))
have (*As*!*i*) · *?ϱ* = (*As*!*i*) · *?ϱi*
by (*smt* (*verit*, *ccfv-SIG*) *comp-apply i map-of-zip-map mk-subst-def nth-mem set-remdups set-rev set-vars-term-list term.set-intros(4) term-subst-eq-conv*)
with *Pfun(1)*[*of As*!*i*] *i Bs-i* **have** (*As*!*i*) · *?ϱ* = *Bs*!*i*
by *fastforce*
}note *As-Bs=this*
with *Bs(2)* **have** *map-ϱ:map* (λ*a*. *a* · *?ϱ*) *As* = *Bs*
by (*simp add*: *map-nth-eq-conv*)
{fix *x* **assume** *x:x* ∈ *vars-term* (*Pfun f As*)
then obtain *i* **where** *i* < *length As x* ∈ *vars-term* (*As*!*i*)
by (*metis term.sel(4) var-imp-var-of-arg*)
with *Pfun(1)*[*of As*!*i*] *Bs-i As-Bs* **have** *σ x* ⊔ *τ x* = *Some* (*?ϱ x*)
using *term-subst-eq-rev* **by** *fastforce*
}
moreover then have *B* = *Pfun f As* · *?ϱ*
using *Pfun(2)* **unfolding** *eval-term.simps join.simps Bs* **using** *map-ϱ* **by** *auto*
moreover then have *match B* (*Pfun f As*) = *Some ?ϱ*
using *match-trivial rho-domain* **by** *blast*
ultimately show *?case* **by** *simp*
next
case (*Prule α As*)
let *?ϱ=mk-subst Var* (*zip* (*vars-distinct* (*Prule α As*)) (*map ?join-var* (*vars-distinct* (*Prule α As*))))
have *rho-domain:subst-domain ?ϱ* ⊆ *vars-term* (*Prule α As*)
by (*smt* (*verit*, *del-insts*) *comp-apply mem-Collect-eq mk-subst-not-mem set-remdups set-rev set-vars-term-list subsetI subst-domain-def*)
from *Prule(2)* **have** *those* (*map2* (⊔) (*map* (λ*a*. *a* · *σ*) *As*) (*map* (λ*a*. *a* · *τ*)

As)) ≠ *None*
unfolding *eval-term.simps join.simps* **using** *option.simps*(*4*) **by** *fastforce*
then obtain *Bs* **where** *Bs*:*those* (*map2* (⊔) (*map* (λ*a*. *a* · σ) *As*) (*map* (λ*a*. *a* · τ) *As*)) = *Some Bs length As* = *length Bs*
using *length-those* **by** *fastforce*
{fix *i* **assume** *i* < *length As*
with *Bs* **have** *Bs-i*:((*As*!*i*) · σ) ⊔ ((*As*!*i*) · τ) = *Some* (*Bs*!*i*)
using *those-some2* **by** *fastforce*
}note *Bs-i*=*this*
{fix *i* **assume** *i*:*i* < *length As*
let *?ϱi*=*mk-subst Var* (*zip* (*vars-distinct* (*As*!*i*)) (*map ?join-var* (*vars-distinct* (*As*!*i*))))
have (*As*!*i*) · *?ϱ* = (*As*!*i*) · *?ϱi*
by (*smt* (*verit*, *ccfv-SIG*) *comp-apply i map-of-zip-map mk-subst-def nth-mem set-remdups set-rev set-vars-term-list term.set-intros*(*4*) *term-subst-eq-conv*)
with *Prule*(*1*)[*of As*!*i*] *i Bs-i* **have** (*As*!*i*) · *?ϱ* = *Bs*!*i*
by *fastforce*
}note *As-Bs*=*this*
with *Bs*(*2*) **have** *map-ϱ*:*map* (λ*a*. *a* · *?ϱ*) *As* = *Bs*
by (*simp add*: *map-nth-eq-conv*)
{fix *x* **assume** *x*:*x* ∈ *vars-term* (*Prule* α *As*)
then obtain *i* **where** *i* < *length As x* ∈ *vars-term* (*As*!*i*)
by (*metis term.sel*(*4*) *var-imp-var-of-arg*)
with *Prule*(*1*)[*of As*!*i*] *Bs-i As-Bs* **have** σ *x* ⊔ τ *x* = *Some* (*?ϱ x*)
using *term-subst-eq-rev* **by** *fastforce*
}
moreover then have *B* = *Prule* α *As* · *?ϱ*
using *Prule*(*2*) **unfolding** *eval-term.simps join.simps Bs* **using** *map-ϱ* **by** *auto*
moreover then have *match B* (*Prule* α *As*) = *Some ?ϱ*
using *match-trivial rho-domain* **by** *blast*
ultimately show *?case* **by** *simp*
qed
then show *?thesis* **by** *blast*
qed

lemma *join-pterm-subst-None*:
fixes *A*::('*f*, '*v*) *pterm*
assumes (*A* · σ) ⊔ (*A* · τ) = *None*
shows ∃ *x* ∈ *vars-term A*. σ *x* ⊔ τ *x* = *None*
using *assms* **proof**(*induct A rule*:*pterm-induct*)
case (*Pfun f As*)
from *Pfun*(*2*) **obtain** *i* **where** *i*:*i* < *length As* (*map2* (⊔) (*map* (λ*s*. *s* · σ) *As*) (*map* (λ*s*. *s* · τ) *As*))!*i* = *None*
unfolding *eval-term.simps join.simps length-map* **using** *those-not-none-xs*
by (*smt* (*verit*) *length-map list-all-length map2-map-map option.case-eq-if option.distinct*(*1*))
then have (*As*!*i* · σ) ⊔ (*As*!*i* · τ) = *None* **by** *simp*
with *Pfun*(*1*) *i*(*1*) **obtain** *x* **where** *x* ∈ *vars-term* (*As*!*i*) **and** σ *x* ⊔ τ *x* = *None*

```
    using nth-mem by blast
  then show ?case using i(1) by auto
next
  case (Prule α As)
  from Prule(2) obtain i where i:i < length As (map2 (⊔) (map (λs. s · σ) As)
(map (λs. s · τ) As))!i = None
    unfolding eval-term.simps join.simps length-map using those-not-none-xs
    by (smt (verit) length-map list-all-length map2-map-map option.case-eq-if op-
tion.distinct(1))
  then have (As!i · σ) ⊔ (As!i · τ) = None by simp
  with Prule(1) i(1) obtain x where x ∈ vars-term (As!i) and σ x ⊔ τ x =
None
    using nth-mem by blast
  then show ?case using i(1) by auto
qed simp

fun mk-subst-from-list :: ('v ⇒ ('f, 'v) term) list ⇒ ('v ⇒ ('f, 'v) term) where
  mk-subst-from-list [] = Var
| mk-subst-from-list (σ # σs) = (λx. case σ x of
      Var x ⇒ mk-subst-from-list σs x
    | t ⇒ t)

lemma join-is-Fun:
  assumes join:A ⊔ B = Some (Pfun f Cs)
  shows ∃ As. A = Pfun f As ∧ length As = length Cs
proof−
  {assume ∃ x. A = Var x
    then obtain x where x:A = Var x by blast
    from join consider B = Var x | ∃ α Bs. B = Prule α Bs
      unfolding x by (metis is-Prule.elims(1) join.simps(1) join.simps(9) op-
tion.distinct(1))
    then have False
    using join unfolding x by(cases) (simp, metis (mono-tags, lifting) Inl-Inr-False
join.simps(6) option.case-eq-if option.distinct(1) option.sel term.inject(2))
  } moreover {assume ∃ α As. A = Prule α As
    then obtain α As where A:A = Prule α As by blast
    from join consider ∃ x. B = Var x | ∃ g Bs. B = Pfun g Bs
      unfolding A by (smt (verit, del-insts) is-Prule.elims(1) join.simps(3) op-
tion.case-eq-if option.distinct(1) option.sel term.inject(2))
    then have False
      using join unfolding A by(cases) (metis (mono-tags, lifting) Inl-Inr-False
join.simps(4,5) option.case-eq-if option.distinct(1) option.sel term.inject(2))+
  }
  ultimately obtain g As where A:A = Pfun g As
    by (meson is-Prule.cases)
  from join have f = g and length As = length Cs unfolding A
    by (smt (verit, ccfv-threshold) Inl-Inr-False Residual-Join-Deletion.join-sym
join.simps(5) join.simps(8) join-fun-fun not-arg-cong-Inr option.case-eq-if option.inject
option.simps(3) pterm-cases term.inject(2))+
```

with *A* **show** *?thesis* **by** *force*
qed

lemma *join-obtain-subst*:
assumes *join*:$A \sqcup B$ = *Some* (*to-pterm t* · σ) **and** *linear-term t*
shows (*to-pterm t*) · *mk-subst Var* (*match-substs* (*to-pterm t*) *A*) = *A*
proof−
from *assms*(*2*) **have** *lin*:*linear-term* (*to-pterm t*)
using *to-pterm-linear* **by** *blast*
have ∀ *p*∈*poss* (*to-pterm t*). ∀ *f ts*. (*to-pterm t*) |- *p* = *Fun f ts* ⟶ (∃ *As*. *length ts* = *length As* ∧ *A* |- *p* = *Fun f As*)
using *assms* **proof**(*induct t arbitrary*: *A B*)
case (*Fun f ts*)
from *Fun*(*2*) **obtain** *As* **where** *A*:*A* = *Pfun f As* **and** *l-As*:*length ts* = *length As*
using *join-is-Fun* **by** *force*
from *Fun*(*2*) **obtain** *Bs* **where** *B*:*B* = *Pfun f Bs* **and** *l-Bs*:*length ts* = *length Bs*
using *join-is-Fun join-sym* **by** (*smt* (*verit*) *eval-term.simps*(*2*) *length-map to-pterm.simps*(*2*))
{**fix** *p g ts*′ **assume** *p*:*p* ∈ *poss* (*to-pterm* (*Fun f ts*)) (*to-pterm* (*Fun f ts*)) |-*p* = *Fun g ts*′
have ∃ *As*′. *length ts*′ = *length As*′ ∧ *A*|-*p* = *Fun g As*′ **proof**(*cases p*)
case *Nil*
from *p*(*2*) **show** *?thesis* **unfolding** *A Nil* **using** *l-As* **by** *force*
next
case (*Cons i p*′)
from *p*(*1*) **have** *i*:*i* < *length ts* **unfolding** *Cons* **by** *simp*
with *p*(*1*) **have** *p*′:*p*′ ∈ *poss* (*ts*!*i*) **unfolding** *Cons*
by (*metis poss-Cons-poss poss-list-sound poss-list-to-pterm term.sel*(*4*))
from *Fun*(*2*) **have** *As*!*i* ⊔ *Bs*!*i* = *Some* (*to-pterm* (*ts*!*i*) · σ)
unfolding *A B to-pterm.simps eval-term.simps* **using** *i l-As l-Bs*
by (*smt* (*verit*, *ccfv-threshold*) *args-poss join-fun-fun local.Cons nth-map p*(*1*) *term.sel*(*4*) *to-pterm.simps*(*2*))
moreover from *Fun*(*3*) *i* **have** *linear-term* (*ts*!*i*) **by** *simp*
ultimately obtain *As*′ **where** *length ts*′ = *length As*′ **and** (*As*!*i*)|-*p*′ = *Fun g As*′
using *Fun*(*1*) *i p*′ **by** (*smt* (*verit*) *local.Cons nth-map nth-mem p*(*2*) *p-in-poss-to-pterm subt-at.simps*(*2*) *to-pterm.simps*(*2*))
with *i l-As* **show** *?thesis* **unfolding** *A Cons* **by** *simp*
qed
}
then show *?case* **by** *simp*
qed *simp*
then show *?thesis* **using** *fun-poss-eq-imp-matches*[*OF lin*] **by** *presburger*
qed

lemma *join-pterm-linear-subst*:
assumes *join*:$A \sqcup B$ = *Some* (*to-pterm t* · σ) **and** *lin*:*linear-term t*

shows $\exists\ \sigma_A\ \sigma_B.\ A = (\textit{to-pterm}\ t \cdot \sigma_A) \wedge B = (\textit{to-pterm}\ t \cdot \sigma_B) \wedge (\forall x \in \textit{vars-term}\ t.\ \sigma_A\ x \sqcup \sigma_B\ x = \textit{Some}\ (\sigma\ x))$
proof−
let $?\sigma_A$=*mk-subst Var* (*match-substs* (*to-pterm t*) *A*)
let $?\sigma_B$=*mk-subst Var* (*match-substs* (*to-pterm t*) *B*)
from *join-obtain-subst*[*OF join lin*] **have** *A*:$A = (\textit{to-pterm}\ t) \cdot ?\sigma_A$ **by** *simp*
from *join lin* **have** *B*:$B = (\textit{to-pterm}\ t) \cdot ?\sigma_B$ **using** *join-sym join-obtain-subst*
by *metis*
from *join-pterm-subst-Some join A B* **obtain** ϱ
where $(\forall x{\in}\textit{vars-term}\ t.\ ?\sigma_A\ x \sqcup ?\sigma_B\ x = \textit{Some}\ (\varrho\ x))$ **and** $\textit{to-pterm}\ t \cdot \sigma = \textit{to-pterm}\ t \cdot \varrho$
by (*metis set-vars-term-list vars-to-pterm*)
then show *?thesis*
by (*smt* (*verit*, *best*) *A B set-vars-term-list term-subst-eq-rev vars-to-pterm*)
qed

context *no-var-lhs*
begin
lemma *join-rule-lhs*:
assumes *wf*:*Prule* α *As* $\in$ *wf-pterm R* **and** *args*:$\forall\ i < \textit{length As}.\ \textit{As}!i \sqcup \textit{Bs}!i \neq \textit{None}$ **and** *l*:*length Bs* = *length As*
shows *Prule* α *As* $\sqcup$ (*to-pterm* (*lhs* α) $\cdot$ $\langle \textit{Bs} \rangle_\alpha$) $\neq$ *None*
proof−
from *wf no-var-lhs* **obtain** *f ts* **where** *lhs*:*lhs* α = *Fun f ts*
by (*metis Inl-inject Term.term.simps*(*2*) *Term.term.simps*(*4*) *case-prodD is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *term.collapse*(*2*) *wf-pterm.simps*)
from *args l* **have** *those* (*map2* ($\sqcup$) *As Bs*) $\neq$ *None*
by (*simp add*: *list-all-length those-not-none-xs*)
with *wf l* **have** *those* (*map2* ($\sqcup$) *As* (*map* ($\langle \textit{Bs} \rangle_\alpha$) (*vars-distinct* (*Fun f ts*)))) $\neq$ *None*
using *apply-lhs-subst-var-rule* **by** (*metis Inl-inject is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *lhs term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
with *lhs-subst-trivial*[*of* α *Bs*] **show** *?thesis*
unfolding *lhs to-pterm.simps eval-term.simps join.simps* **by** *force*
qed
end

3.2.1 N-Fold Join

We define a function to recursively join a list of n proof terms. Since each individual join produces a (('*f*, '*v*) *prule* + '*f*, '*v*) *Term.term option* we first introduce the following helper function.

fun *join-opt* :: ('*f*, '*v*) *pterm* $\Rightarrow$ ('*f*, '*v*) *pterm option* $\Rightarrow$ ('*f*, '*v*) *pterm option*
where
join-opt A (*Some B*) = *A* $\sqcup$ *B*
| *join-opt* - - = *None*

fun *join-list* :: ('*f*, '*v*) *pterm list* $\Rightarrow$ ('*f*,'*v*) *pterm option* ($\bigsqcup$)
where

```
  join-list [] = None
| join-list (A # []) = Some A
| join-list (A # As) = join-opt A (join-list As)

context left-lin-no-var-lhs
begin

lemma join-var-rule:
  assumes to-rule α ∈ R
  shows Var x ⊔ Prule α As = None
proof−
  from assms obtain f ts where lhs α = Fun f ts
    using no-var-lhs by fastforce
  then show ?thesis
   by (metis (no-types, lifting) Residual-Join-Deletion.join-sym eval-term.simps(2)
join.simps(4) match-lhs-subst option.case-eq-if option.exhaust term.distinct(1) to-pterm.simps(2))
qed

lemma var-join:
  assumes Var x ⊔ B = Some C and B ∈ wf-pterm R
  shows B = Var x ∧ C = Var x
  using assms proof(cases B)
  case (Var y)
  with assms(1) show ?thesis
    by (metis join.simps(1) option.sel option.simps(3))
next
  case (Prule α As)
  with assms show ?thesis
     by (metis Residual-Join-Deletion.join-sym Term.term.simps(4) case-prodD
co-initial-Var is-VarI join.simps(9)
     no-var-lhs.no-var-lhs no-var-lhs-axioms option.distinct(1) source-join sum.inject(1)
term.inject(2) wf-pterm.simps)
qed simp

lemma fun-join:
  assumes Pfun f As ⊔ B = Some C
  shows (∃ g Bs. B = Pfun g Bs) ∨ (∃ α Bs. B = Prule α Bs)
  using assms by(cases B) (simp-all)

lemma rule-join:
  assumes Prule α As ⊔ B = Some C and Prule α As ∈ wf-pterm R
  shows (∃ g Bs. B = Pfun g Bs) ∨ (∃ β Bs. B = Prule β Bs)
  using assms proof(cases B)
  case (Var x)
  from assms have False unfolding Var
    by (metis Residual-Join-Deletion.join-sym term.distinct(1) var-join)
  then show ?thesis by simp
qed simp-all
```

Associativity of join is currently not used in any proofs. But it is still a

valuable result, hence included here.

lemma *join-opt-assoc*:
assumes $A \in$ *wf-pterm R* $B \in$ *wf-pterm R* $C \in$ *wf-pterm R*
shows *join-opt A* (*B* ⊔ *C*) = *join-opt C* (*A* ⊔ *B*)
using *assms* **proof**(*induct A arbitrary*:*B C rule*:*subterm-induct*)
case (*subterm A*)
from *subterm*(*2*) **show** *?case* **proof**(*cases A rule*:*wf-pterm.cases*[*case-names VarA FunA RuleA*])
case (*VarA x*)
with *subterm*(*3*) **show** *?thesis* **proof**(*cases B rule*:*wf-pterm.cases*[*case-names VarB FunB RuleB*])
case (*VarB y*)
show *?thesis* **proof**(*cases x* = *y*)
case *True*
then have *AB*:*A* ⊔ *B* = *Some* (*Var y*) **unfolding** *VarA VarB* **by** *simp*
with *subterm*(*4*) *VarA VarB* **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*VarC z*)
with *AB VarA VarB* ‹*x* = *y*› **show** *?thesis* **by**(*cases y* = *z*) *simp-all*
next
case (*RuleC* α *As*)
then have (*Var y*) ⊔ *C* = *None*
using *join-var-rule* **by** *presburger*
then show *?thesis* **unfolding** *AB* **unfolding** *VarA VarB* **by** (*simp add*:*join-sym*)
qed *simp*
next
case *False*
then have *AB*:*A* ⊔ *B* = *None* **unfolding** *VarA VarB* **by** *simp*
with *subterm*(*4*) *VarA VarB* **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*VarC z*)
with *AB VarA VarB* ‹*x* ≠ *y*› **show** *?thesis* **by**(*cases y* = *z*) *simp-all*
next
case (*RuleC* α *As*)
then have (*Var y*) ⊔ *C* = *None*
using *join-var-rule* **by** *presburger*
then show *?thesis* **unfolding** *AB* **unfolding** *VarA VarB* **by** (*simp add*:*join-sym*)
qed *simp*
qed
next
case (*FunB Bs f*)
then have *AB*:*A* ⊔ *B* = *None*
unfolding *VarA* **by** *simp*
with *subterm*(*4*) *VarA* **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*VarC z*)
with *AB VarA FunB* **show** *?thesis* **by**(*cases x* = *z*) *simp-all*

```
      next
        case (FunC Cs g)
        from AB VarA show ?thesis proof(cases B ⊔ C)
          case (Some BC)
          then obtain BCs where BC = Pfun f BCs
            by (metis FunB(1) FunC(1) join-fun-fun)
          then show ?thesis unfolding AB unfolding VarA Some by simp
        qed simp
      next
        case (RuleC α Cs)
        from AB VarA show ?thesis proof(cases B ⊔ C)
          case (Some BC)
          then obtain BCs where BC = Prule α BCs
         by (metis FunB(1) Residual-Join-Deletion.join-sym RuleC(1) join-rule-fun
subterm.prems(3))
          then have Var x ⊔ BC = None
            using RuleC(2) join-var-rule by presburger
          then show ?thesis unfolding AB unfolding VarA Some by simp
        qed simp
      qed
    next
      case (RuleB α Bs)
      then have AB:A ⊔ B = None
        using VarA join-var-rule by presburger
    with subterm(4) VarA show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
        case (VarC z)
        with RuleB have B ⊔ C = None
          using join-var-rule join-sym by metis
        with AB show ?thesis by simp
      next
        case (FunC Cs f)
        from AB VarA show ?thesis proof(cases B ⊔ C)
          case (Some BC)
          then obtain BCs where BC = Prule α BCs
            by (metis FunC(1) RuleB(1) join-rule-fun subterm.prems(2))
          then have Var x ⊔ BC = None
            using RuleB(2) join-var-rule by presburger
          then show ?thesis unfolding AB unfolding VarA Some by simp
        qed simp
      next
        case (RuleC β Cs)
        from AB VarA show ?thesis proof(cases B ⊔ C)
          case (Some BC)
          then obtain BCs where BC = Prule α BCs
         using RuleB(1) RuleC(1) join-rule-rule subterm.prems(2) subterm.prems(3)
by blast
          then have Var x ⊔ BC = None
            using RuleB(2) join-var-rule by presburger
```

```
        then show ?thesis unfolding AB unfolding VarA Some by simp
       qed simp
      qed
    qed
  next
    case (FunA As f)
    from subterm(3) show ?thesis proof(cases B rule:wf-pterm.cases[case-names
VarB FunB RuleB])
      case (VarB x)
      then show ?thesis
       by (metis FunA(1) join.simps(1) join.simps(8) join.simps(9) join-opt.elims
join-opt.simps(2) join-var-rule option.sel subterm.prems(3) wf-pterm.simps)
    next
      case (FunB Bs g)
      then show ?thesis proof(cases A ⊔ B)
        case None
      with subterm(4) FunB show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
        case (FunC Cs h)
        from None show ?thesis proof(cases B ⊔ C)
          case (Some BC)
          then have gh:g = h and l-B-C:length Bs = length Cs
            unfolding FunB FunC by (meson join-fun-fun)+
        from Some obtain BCs where BC:BC = Pfun g BCs and l-BC-B:length
BCs = length Bs
            and args-BC:(∀ i<length Bs. Bs!i ⊔ Cs!i = Some (BCs ! i))
            unfolding FunC FunB using join-fun-fun by force
          {assume fg:f = g and l-A-B:length As = length Bs
            {fix i assume i:i < length As
              with subterm(1) FunA FunB(2) FunC(2) args-BC l-A-B l-B-C
                have join-opt (As!i) ((Bs!i) ⊔ (Cs!i)) = join-opt (Cs!i) ((As!i) ⊔
(Bs!i))
                  by (metis nth-mem supt.arg)
            } note IH=this
            from fg l-A-B None have those (map2 (⊔) As Bs) = None
             unfolding FunB FunA by (smt (verit) join.simps(2) option.case-eq-if
option.distinct(1))
             then obtain i where i:i < length (map2 (⊔) As Bs) (map2 (⊔) As
Bs)!i = None
              using those-not-none-xs list-all-length by blast
            with l-A-B have A-B-i:(As!i) ⊔ (Bs!i) = None by simp
             with IH i(1) l-B-C have As!i ⊔ BCs!i = None using args-BC by
fastforce
           with i(1) l-BC-B l-B-C l-A-B have those (map2 (⊔) As BCs) = None
             using list-all-length those-some2 by fastforce
          }
          then show ?thesis
            using l-B-C l-BC-B FunA FunB FunC BC gh None Some by auto
        qed simp
```

next
case (*RuleC* α *Cs*)
from *None* **show** *?thesis* **proof**(*cases B* $\sqcup$ *C*)
case (*Some BC*)
then obtain *BCs* τ **where** τ:*match B* (*to-pterm* (*lhs* α)) = *Some* τ **and** *BC*:*BC* = *Prule* α *BCs*
and *l-BCs*:*length BCs* = *length Cs* **and** *args-BC*:$\forall$ *i* < *length Cs*. *Cs*!*i* $\sqcup$ τ (*var-rule* α ! *i*) = *Some* (*BCs* ! *i*)
by (*metis FunB*(*1*) *join-sym RuleC*(*1*) *join-rule-fun subterm.prems*(*3*))

with *None Some FunA* **show** *?thesis* **proof**(*cases match A* (*to-pterm* (*lhs* α)))
case (*Some* σ)
with τ *None* **obtain** *x* **where** *x*:*x* $\in$ *vars-term* (*lhs* α) σ *x* $\sqcup$ τ *x* = *None*
using *join-pterm-subst-None* **by** (*metis lhs-subst-trivial match-lhs-subst option.sel set-vars-term-list vars-to-pterm*)
then obtain *i* **where** *i*:*i* < *length* (*var-rule* α) *var-rule* α ! *i* = *x*
by (*metis RuleC*(*2*) *case-prodD in-set-idx left-lin left-linear-trs-def linear-term-var-vars-term-list set-vars-term-list*)
have *subt*:*A* $\rhd$ σ *x* **proof**−
obtain *g ts* **where** *lhs*:*lhs* α = *Fun g ts*
using *RuleC*(*2*) *no-var-lhs* **by** *fastforce*
from *Some i* **show** *?thesis*
unfolding *lhs* **by** (*metis* (*no-types*, *lifting*) *lhs match-matches set-vars-term-list subst-image-subterm to-pterm.simps*(*2*) *vars-to-pterm x*(*1*))
qed
have *wf-*τ*-x*:τ *x* $\in$ *wf-pterm R*
using *FunB* τ *i* **by** (*metis match-well-def subterm.prems*(*2*) *vars-to-pterm*)
have *IH*:*join-opt* (σ *x*) (τ *x* $\sqcup$ (*Cs* ! *i*)) = *join-opt* (*Cs*!*i*) (σ *x* $\sqcup$ τ *x*)
using *subterm*(*1*) *RuleC*(*3*,*4*) *i wf-*τ*-x* **by** (*metis Some match-well-def nth-mem subt subterm.prems*(*1*) *vars-to-pterm*)
have τ *x* $\sqcup$ (*Cs* ! *i*) = *Some* (*BCs* ! *i*)
using *args-BC i* **by** (*metis Residual-Join-Deletion.join-sym RuleC*(*3*))

with *IH x*(*2*) **have** (σ *x*) $\sqcup$ (*BCs* ! *i*) = *None* **by** *simp*
then have (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *BCs*) ! *i* = *None*
using *l-BCs i* **by** (*simp add*: *RuleC*(*3*))
then have *those* (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *BCs*) = *None*
using *l-BCs i* **by** (*metis* (*no-types*, *opaque-lifting*) *RuleC*(*3*) *length-map length-zip min.idem not-Some-eq nth-mem those-not-none-x*)
then have *A* $\sqcup$ *BC* = *None*
using *Some i* **unfolding** *BC FunA join.simps* **by** *simp*
then show *?thesis*
unfolding *None* ‹*B* $\sqcup$ *C* = *Some BC*› **by** *auto*
qed *simp*
qed *simp*
qed *simp*

```
    next
      case (Some AB)
      then have fg:f = g and l-A-B:length As = length Bs
        unfolding FunA FunB by (meson join-fun-fun)+
       from Some obtain ABs where AB:AB = Pfun f ABs and l-AB-A:length
ABs = length As
        and args-AB:(∀ i<length Bs. As!i ⊔  Bs!i = Some (ABs ! i))
        unfolding FunA FunB using join-fun-fun by force
     from subterm(4) show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
        case (VarC x)
       show ?thesis unfolding Some AB unfolding FunA FunB VarC by simp
      next
        case (FunC Cs h)
        show ?thesis proof(cases B ⊔ C)
          case None
          {assume gh:g = h and l-B-C:length Bs = length Cs
            {fix i assume i:i < length As
              with subterm(1) FunA FunB(2) FunC(2) args-AB l-A-B l-B-C
                have join-opt (As!i) ((Bs!i) ⊔ (Cs!i)) = join-opt (Cs!i) ((As!i) ⊔
(Bs!i))
                  by (metis nth-mem supt.arg)
            } note IH=this
            from gh l-B-C None have those (map2 (⊔) Bs Cs) = None
            unfolding FunB FunC by (smt (verit) join.simps(2) option.case-eq-if
option.distinct(1))
             then obtain i where i:i < length (map2 (⊔) Bs Cs) (map2 (⊔) Bs
Cs)!i = None
              using those-not-none-xs list-all-length by blast
            with l-B-C have B-C-i:(Bs!i) ⊔ (Cs!i) = None by simp
              with IH i(1) l-A-B have Cs!i ⊔ ABs!i = None using args-AB by
fastforce
           with i(1) l-AB-A l-B-C l-A-B have those (map2 (⊔) Cs ABs) = None
              using list-all-length those-some2 by fastforce
          }
          then show ?thesis
            using l-A-B l-AB-A FunA FunB FunC AB fg None Some by auto
        next
          case (Some BC)
          then have gh:g = h and l-B-C:length Bs = length Cs
            unfolding FunB FunC by(meson join-fun-fun)+
        from Some obtain BCs where BC:BC = Pfun g BCs and l-BC-B:length
BCs = length Bs
            and args-BC:(∀ i<length Cs. Bs!i ⊔ Cs!i = Some (BCs ! i))
            unfolding FunB FunC using join-fun-fun by force
          {fix i assume i:i < length As
            with subterm(1) FunA FunB(2) FunC(2) args-AB l-A-B l-B-C
            have join-opt (As!i) ((Bs!i) ⊔ (Cs!i)) = (Cs!i) ⊔ (ABs!i)
              by (metis join-opt.simps(1) nth-mem supt.arg)
```

with *args-BC i l-A-B l-B-C* **have** $(As!i) \sqcup (BCs!i) = (Cs!i) \sqcup (ABs!i)$ **by** *simp*
} note *IH=this*
then have *those* (*map2* $(\sqcup)$ *As BCs*) = *those* (*map2* $(\sqcup)$ *Cs ABs*)
by (*smt* (*verit*, *del-insts*) *l-AB-A l-A-B l-BC-B l-B-C length-zip map-equality-iff min-less-iff-conj nth-zip old.prod.case*)
then show *?thesis* **unfolding** *Some BC* ‹$A \sqcup B$ = *Some AB*› *AB* **unfolding** *gh FunA FunC fg join-opt.simps* **using** *l-BC-B l-AB-A l-A-B l-B-C* **by** *simp*
qed
next
case (*RuleC* α *Cs*)
from *RuleC*(*2*) **have** *lin*:*linear-term* (*lhs* α)
using *left-lin left-linear-trs-def* **by** *fastforce*
from *RuleC*(*2*) **obtain** f' *ts* **where** *lhs*:*lhs* α = *Fun* f' *ts*
using *no-var-lhs* **by** *fastforce*
consider *match A* (*to-pterm* (*lhs* α)) = *None* | *match B* (*to-pterm* (*lhs* α)) = *None*
| (*matches*) *match A* (*to-pterm* (*lhs* α)) $\neq$ *None* $\wedge$ *match B* (*to-pterm* (*lhs* α)) $\neq$ *None* **by** *linarith*
then show *?thesis* **proof**(*cases*)
case *1*
then have *match*:*match AB* (*to-pterm* (*lhs* α)) = *None*
using *lin* **by** (*smt* (*verit*, *ccfv-threshold*) *Some join-pterm-linear-subst match-complete' match-matches not-Some-eq*)
then have $C \sqcup AB$ = *None*
unfolding *RuleC AB join.simps* **by** *simp*
moreover have *join-opt A* $(B \sqcup C)$ = *None* **proof**−
consider $(\exists\, BCs.\ B \sqcup C$ = *Some* (*Prule* α *BCs*)) | $B \sqcup C$ = *None*
unfolding *FunB RuleC join.simps* **by** (*metis* (*no-types*, *lifting*) *option.case-eq-if*)
then show *?thesis* **using** *1 FunA*(*1*) **by**(*cases*) (*force*, *simp*)
qed
ultimately show *?thesis* **using** *Some* **by** *simp*
next
case *2*
then have *match*:*match AB* (*to-pterm* (*lhs* α)) = *None*
using *lin* **by** (*smt* (*verit*, *ccfv-threshold*) *Some join-pterm-linear-subst match-complete' match-matches not-Some-eq*)
then have $C \sqcup AB$ = *None*
unfolding *RuleC AB join.simps* **by** *simp*
moreover from *2* **have** $B \sqcup C$ = *None*
unfolding *FunB RuleC join.simps* **by** *simp*
ultimately show *?thesis* **using** *Some* **by** *simp*
next
case *matches*
from *matches* **obtain** σ **where** *sigma*:*match A* (*to-pterm* (*lhs* α)) = *Some* σ **by** *force*
from *matches* **obtain** τ **where** *tau*:*match B* (*to-pterm* (*lhs* α)) = *Some*

τ **by** *force*
from *sigma tau* **obtain** ϱ **where** *rho*:($\forall$ x∈*vars-term* (*to-pterm* (*lhs* α)). σ x ⊔ τ x = *Some* (ϱ x))
and *AB-rho*:*AB* = (*to-pterm* (*lhs* α)) · ϱ **and** *match-rho*:*match AB* (*to-pterm* (*lhs* α)) = *Some* ϱ
using *join-pterm-subst-Some match-matches Some* **by** *blast*
{fix i **assume** i:i < *length Cs*
with *sigma RuleC*(*3*) **have** (*map* σ (*var-rule* α))!i ◁ A
using *lhs* **by** (*smt* (*verit*, *ccfv-threshold*) *lin linear-term-var-vars-term-list match-matches nth-map nth-mem set-vars-term-list subst-image-subterm to-pterm.simps*(*2*) *vars-to-pterm*)
moreover have (*map* σ (*var-rule* α))!i ∈ *wf-pterm R*
using i *match-well-def*[*OF subterm*(*2*) *sigma*] *RuleC*(*3*) **by** (*simp add*: *vars-to-pterm*)
moreover have (*map* τ (*var-rule* α))!i ∈ *wf-pterm R*
using i *match-well-def*[*OF subterm*(*3*) *tau*] *RuleC*(*3*) **by** (*simp add*: *vars-to-pterm*)
ultimately have *join-opt* (*map* σ (*var-rule* α) ! i) (*map* τ (*var-rule* α) ! i ⊔ *Cs*!i) = *Cs* ! i ⊔ *map* ϱ (*var-rule* α) ! i
using *subterm*(*1*) *RuleC*(*3*,*4*) i **by** (*smt* (*verit*, *best*) *join-opt.simps*(*1*) *lin linear-term-var-vars-term-list nth-map nth-mem rho set-vars-term-list vars-to-pterm*)

}note *IH*=*this*
show *?thesis* **proof**(*cases those* (*map2* (⊔) (*map* τ (*var-rule* α)) *Cs*))
case *None*
then obtain i **where** i:i < *length Cs* (*map* τ (*var-rule* α))!i ⊔ *Cs*!i = *None*
using *those-not-none-xs* **by** (*smt* (*verit*) *length-map length-zip list-all-length map-nth-eq-conv min-less-iff-conj nth-zip old.prod.case*)
with *IH* **have** *Cs*!i ⊔ *map* ϱ (*var-rule* α) ! i = *None* **by** *force*
with i *RuleC*(*3*) **have** i < *length* (*map2* (⊔) *Cs* (*map* ϱ (*var-rule* α))) (*map2* (⊔) *Cs* (*map* ϱ (*var-rule* α))) ! i = *None* **by** *simp-all*
then have *those* (*map2* (⊔) *Cs* (*map* ϱ (*var-rule* α))) = *None*
by (*metis nth-mem option.exhaust those-not-none-x*)
with *None* **show** *?thesis* **unfolding** *Some* **unfolding** *FunA FunB RuleC join.simps tau*[*unfolded FunB*] **using** *AB match-rho* **by** *auto*
next
case (*Some BCs*)
then have *BC*:*B* ⊔ *C* = *Some* (*Prule* α *BCs*)
unfolding *FunB RuleC join.simps tau*[*unfolded FunB*] **by** *simp*
from *Some* **have** *l-BCs*:*length BCs* = *length Cs*
using *RuleC*(*3*) *length-those* **by** *fastforce*
{fix i **assume** i < *length Cs*
with *Some IH* **have** (*map* σ (*var-rule* α)) ! i ⊔ *BCs* ! i = *Cs* ! i ⊔ (*map* ϱ (*var-rule* α)) ! i
using *RuleC*(*3*) *those-some2* **by** *fastforce*
}
then have *map2* (⊔) (*map* σ (*var-rule* α)) *BCs* = *map2* (⊔) *Cs* (*map* ϱ (*var-rule* α))

```
            using l-BCs by (simp add: RuleC(3) map-eq-conv')
          then show ?thesis unfolding BC ‹A ⊔ B = Some AB› unfolding FunA
FunB RuleC join-opt.simps join.simps sigma[unfolded FunA] using AB match-rho
by auto
         qed
        qed
       qed
      qed
    next
      case (RuleB α Bs)
      from RuleB(2) have lin:linear-term (lhs α)
        using left-lin left-linear-trs-def by fastforce
      from RuleB(2) obtain f' ts where lhs:lhs α = Fun f' ts
        using no-var-lhs by fastforce
      show ?thesis proof(cases A ⊔ B)
        case None
      with subterm(4) RuleB show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
        case (VarC x)
        with subterm(4) RuleB FunA show ?thesis
          by (metis None join-sym join-opt.simps(2) join-var-rule)
       next
        case (FunC Cs h)
        from None show ?thesis proof(cases B ⊔ C)
          case (Some BC)
            obtain BCs τ where τ:match C (to-pterm (lhs α)) = Some τ and
BC:BC = Prule α BCs
            and l-BCs:length BCs = length Bs and args-BC:∀ i < length Bs. Bs!i
⊔ τ (var-rule α ! i) = Some (BCs ! i)
          using join-rule-fun[OF Some[unfolded RuleB FunC] subterm(3)[unfolded
RuleB]] FunC(1) by blast
           with None Some FunA show ?thesis proof(cases match A (to-pterm
(lhs α)))
            case (Some σ)
            from None obtain i where i:i < length (var-rule α) map2 (⊔) (map
σ (var-rule α)) Bs ! i = None
              unfolding FunA RuleB join.simps Some[unfolded FunA] option.case
                by (smt (verit, ccfv-threshold) length-map length-zip list-all-length
min-less-iff-conj option.case-eq-if option.distinct(1) those-not-none-xs)
            let ?x=var-rule α ! i
            have subt:A ▷ σ ?x using lhs i Some
          by (smt (verit, ccfv-SIG) lin linear-term-var-vars-term-list match-matches
nth-mem set-vars-term-list subst-image-subterm to-pterm.simps(2) vars-to-pterm)

            have wf-τ-x:τ ?x ∈ wf-pterm R
              using subterm(4) τ i(1) by (metis match-well-def vars-to-pterm)
            have IH:join-opt (σ ?x) (Bs ! i ⊔ τ ?x) = join-opt (τ ?x) (σ ?x ⊔ Bs
! i)
                  using subterm(1) i wf-τ-x by (metis RuleB(3) RuleB(4) Some
```

match-well-def nth-mem subt subterm.prems(1) vars-to-pterm)
have (*Bs* ! *i* ⊔ *τ ?x*) = *Some* (*BCs* ! *i*)
using *args-BC i RuleB*(*3*) **by** *auto*
with *IH i* **have** (*σ ?x*) ⊔ (*BCs* ! *i*) = *None*
by (*simp add*: *RuleB*(*3*))
then have (*map2* (⊔) (*map σ* (*var-rule α*)) *BCs*) ! *i* = *None*
using *l-BCs i* **by** (*simp add*: *RuleB*(*3*))
then have *those* (*map2* (⊔) (*map σ* (*var-rule α*)) *BCs*) = *None*
using *l-BCs i* **by** (*metis* (*no-types*, *opaque-lifting*) *RuleB*(*3*) *length-map length-zip min.idem nth-mem option.exhaust those-not-none-x*)
then have *A* ⊔ *BC* = *None*
using *Some i* **unfolding** *BC FunA join.simps* **by** *simp*
then show *?thesis*
unfolding *None* ‹*B* ⊔ *C* = *Some BC*› **by** *auto*
qed *simp*
qed *simp*
next
case (*RuleC β Cs*)
from *None* **show** *?thesis* **proof**(*cases B* ⊔ *C*)
case (*Some BC*)
then have *αβ*:*α* = *β* **and** *l-B-C*:*length Bs* = *length Cs*
using *join-rule-rule*[*OF Some*[*unfolded RuleB RuleC*] *subterm*(*3*,*4*)[*unfolded RuleB RuleC*]] **by** *simp*+
from *Some* **obtain** *BCs* **where** *BC*:*BC* = *Prule α BCs* **and** *l-BC-B*:*length BCs* = *length Bs*
and *args-BC*:(∀ *i*<*length Cs*. *Bs*!*i* ⊔ *Cs*!*i* = *Some* (*BCs* ! *i*))
using *join-rule-rule*[*OF Some*[*unfolded RuleB RuleC*] *subterm*(*3*,*4*)[*unfolded RuleB RuleC*]] **by** *force*
from *Some FunA RuleB BC* **show** *?thesis* **proof**(*cases match A* (*to-pterm* (*lhs α*)))
case (*Some σ*)
from *None* **obtain** *i* **where** *i*:*i* < *length* (*var-rule α*) *map2* (⊔) (*map σ* (*var-rule α*)) *Bs* ! *i* = *None*
unfolding *FunA RuleB join.simps Some*[*unfolded FunA*] *option.case*
by (*smt* (*verit*, *ccfv-threshold*) *length-map length-zip list-all-length min-less-iff-conj option.case-eq-if option.distinct*(*1*) *those-not-none-xs*)
let *?x*=*var-rule α* ! *i*
have *subt*:*A* ▷ *σ ?x* **using** *lhs i Some*
by (*smt* (*verit*, *ccfv-SIG*) *lin linear-term-var-vars-term-list match-matches nth-mem set-vars-term-list subst-image-subterm to-pterm.simps*(*2*) *vars-to-pterm*)

have *IH*:*join-opt* (*σ ?x*) (*Bs* ! *i* ⊔ *Cs* ! *i*) = *join-opt* (*Cs* ! *i*) (*σ ?x* ⊔ *Bs* ! *i*)
using *subterm*(*1*) *i RuleC* **by** (*metis RuleB*(*3*) *RuleB*(*4*) *Some αβ match-well-def nth-mem subt subterm.prems*(*1*) *vars-to-pterm*)
from *IH i* **have** (*σ ?x*) ⊔ (*BCs* ! *i*) = *None*
using *RuleB*(*3*) *args-BC l-B-C* **by** *auto*
then have (*map2* (⊔) (*map σ* (*var-rule α*)) *BCs*) ! *i* = *None*
using *RuleB*(*3*) *i*(*1*) *l-BC-B* **by** *force*

```
          then have those (map2 (⊔) (map σ (var-rule α)) BCs) = None
           by (metis (no-types, opaque-lifting) RuleB(3) i(1) l-BC-B length-map
length-zip min.idem not-Some-eq nth-mem those-not-none-x)
          then have A ⊔ BC = None
            using Some i unfolding BC FunA join.simps by simp
          then show ?thesis
            unfolding None ‹B ⊔ C = Some BC› by auto
        qed simp
      qed simp
    qed
  next
    case (Some AB)
    then obtain σ ABs where sigma:match A (to-pterm (lhs α)) = Some σ
      and AB:AB = Prule α ABs and l-ABs:length ABs = length Bs
     and args-AB:(∀ i<length Bs. σ (var-rule α ! i) ⊔ Bs ! i = Some (ABs ! i))
    unfolding FunA RuleB using join-sym join-rule-fun subterm(2,3)[unfolded
FunA RuleB] RuleB(3) by (smt (verit, del-insts))
    {fix i assume i:i < length Bs
     with sigma RuleB(3) have (map σ (var-rule α))!i ◁ A
      using lhs by (smt (verit, ccfv-threshold) lin linear-term-var-vars-term-list
match-matches nth-map nth-mem set-vars-term-list subst-image-subterm to-pterm.simps(2)
vars-to-pterm)
    }note A-sub=this
  from subterm(4) show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
    case (VarC x)
    have match (Var x) (to-pterm (lhs α)) = None
        unfolding lhs to-pterm.simps using match-matches not-None-eq by
fastforce
    then show ?thesis
      unfolding Some unfolding RuleB VarC AB by simp
  next
    case (FunC Cs g)
    show ?thesis proof(cases match C (to-pterm (lhs α)))
      case None
      then have B ⊔ C = None
        unfolding RuleB FunC by simp
      moreover from None have AB ⊔ C = None
        unfolding AB FunC by simp
      ultimately show ?thesis
        unfolding Some by (simp add: join-sym)
    next
      case (Some τ)
      {fix i assume i:i < length Bs
        have (map σ (var-rule α))!i ∈ wf-pterm R
          using i match-well-def[OF subterm(2) sigma] RuleB(3) by (simp
add: vars-to-pterm)
        moreover have (map τ (var-rule α))!i ∈ wf-pterm R
          using i match-well-def[OF subterm(4) Some] RuleB(3) by (simp
```

```
add: vars-to-pterm)
            ultimately have join-opt (map σ (var-rule α) ! i) (Bs!i ⊔ map τ
(var-rule α) ! i) = ABs ! i ⊔ map τ (var-rule α) ! i
              using subterm(1) RuleB(3,4) i args-AB A-sub join-sym
              by (smt (verit, ccfv-SIG) join-opt.simps(1) nth-map nth-mem)
          }note IH=this
          show ?thesis proof(cases those (map2 (⊔) Bs (map τ (var-rule α))))
            case None
           then obtain i where i:i < length Bs Bs!i ⊔ (map τ (var-rule α))!i =
None
                  using those-not-none-xs by (smt (verit) length-map length-zip
list-all-length map-nth-eq-conv min-less-iff-conj nth-zip old.prod.case)
            with IH have map τ (var-rule α) ! i ⊔ ABs ! i = None
              using join-sym by (metis join-opt.simps(2))
            with i RuleB(3) l-ABs have i < length (map2 (⊔) (map τ (var-rule
α)) ABs) (map2 (⊔) (map τ (var-rule α)) ABs) ! i = None by simp-all
            then have those (map2 (⊔) (map τ (var-rule α)) ABs) = None
              by (metis nth-mem option.exhaust those-not-none-x)
            with None show ?thesis
                  unfolding ‹A ⊔ B = Some AB› AB unfolding RuleB FunC
join-opt.simps join.simps Some[unfolded FunC] option.case None by simp
          next
            case (Some BCs)
            then have BC:B ⊔ C = Some (Prule α BCs)
               unfolding RuleB FunC join.simps ‹match C (to-pterm (lhs α)) =
Some τ›[unfolded FunC] by simp
            from Some have l-BCs:length BCs = length Bs
              using RuleB(3) length-those by fastforce
            {fix i assume i < length Bs
               with Some IH have (map σ (var-rule α)) ! i ⊔ BCs!i = (map τ
(var-rule α)) ! i ⊔ ABs ! i
                 using RuleB(3) those-some2 join-sym by fastforce
            }
           then have map2 (⊔) (map σ (var-rule α)) BCs = map2 (⊔) (map τ
(var-rule α)) ABs
              using l-BCs l-ABs by (simp add: RuleB(3) map-eq-conv')
         then show ?thesis unfolding BC ‹A ⊔ B = Some AB› AB unfolding
FunA RuleB FunC AB join-opt.simps join.simps sigma[unfolded FunA]
             ‹match C (to-pterm (lhs α)) = Some τ›[unfolded FunC] option.case
by simp
        qed
      qed
    next
      case (RuleC β Cs)
      show ?thesis proof(cases α = β)
        case True
        with RuleB(3) RuleC(3) have l-Bs-Cs:length Bs = length Cs by simp
        {fix i assume i:i < length Bs
          have (map σ (var-rule α))!i ∈ wf-pterm R
```

```
            using i match-well-def[OF subterm(2) sigma] RuleB(3) by (simp
add: vars-to-pterm)
          then have join-opt (map σ (var-rule α) ! i) (Bs!i ⊔ Cs ! i) =  Cs ! i
⊔ ABs ! i
            using subterm(1) RuleB(3,4) RuleC(3,4) i args-AB A-sub True by
simp
        }note IH=this
        show ?thesis proof(cases those (map2 (⊔) Bs Cs))
          case None
          then obtain i where i:i < length Bs Bs!i ⊔ Cs!i = None
              using those-not-none-xs by (smt (verit) length-map length-zip
list-all-length map-nth-eq-conv min-less-iff-conj nth-zip old.prod.case)
          with IH have Cs ! i ⊔ ABs ! i = None by force
           with i RuleB(3) l-ABs l-Bs-Cs have i < length (map2 (⊔) Cs ABs)
(map2 (⊔) Cs ABs) ! i = None by simp-all
          then have those (map2 (⊔) Cs ABs) = None
            by (metis nth-mem option.exhaust those-not-none-x)
        with None show ?thesis unfolding ‹A ⊔ B = Some AB› AB unfolding
RuleB RuleC by simp
        next
          case (Some BCs)
          then have BC:B ⊔ C = Some (Prule α BCs)
            unfolding RuleB RuleC True by simp
          from Some have l-BCs:length BCs = length Bs
            using l-Bs-Cs length-those by fastforce
          {fix i assume i < length Bs
             with Some IH have (map σ (var-rule α)) ! i ⊔ BCs!i = Cs ! i ⊔
ABs ! i
              using those-some2 l-Bs-Cs by fastforce
          }
         then have map2 (⊔) (map σ (var-rule α)) BCs  = map2 (⊔) Cs ABs
              using l-Bs-Cs RuleB(3) l-ABs l-BCs by (simp add: RuleC(3)
map-eq-conv')
          then show ?thesis
             unfolding BC ‹A ⊔ B = Some AB› AB unfolding FunA RuleB
RuleC join-opt.simps join.simps sigma[unfolded FunA] unfolding True by simp
        qed
      next
        case False
        then show ?thesis
        unfolding ‹A ⊔ B = Some AB› unfolding RuleB RuleC AB join.simps
by simp
      qed
    qed
  qed
 next
  case (RuleA α As)
  from RuleA(2) have lin:linear-term (lhs α)
```

```
    using left-lin left-linear-trs-def by fastforce
  from RuleA(2) obtain f' ts where lhs:lhs α = Fun f' ts
    using no-var-lhs by fastforce
 from subterm(3,2) show ?thesis proof(cases B rule:wf-pterm.cases[case-names
VarB FunB RuleB])
    case (VarB x)
    have match (Var x) (to-pterm (lhs α)) = None
     unfolding lhs using match-matches not-Some-eq by fastforce
    then show ?thesis unfolding RuleA VarB
     by (metis join-sym RuleA(2) join.simps(1) join.simps(9) join-opt.simps(1)
join-opt.simps(2)
       left-lin-no-var-lhs.join-var-rule left-lin-no-var-lhs-axioms subterm.prems(3)
wf-pterm.simps)
  next
    case (FunB Bs f)
    show ?thesis proof(cases A ⊔ B)
     case None
   with subterm(4) FunB show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
      case (VarC x)
      with subterm(4) FunB RuleA None show ?thesis
       by auto
     next
      case (FunC Cs h)
      from None show ?thesis proof(cases B ⊔ C)
       case (Some BC)
       have fh:f = h and l-B-C:length Bs = length Cs
        using join-fun-fun[OF Some[unfolded FunB FunC]] by simp+
       obtain BCs where BC:BC = Pfun f BCs and l-BC-B:length BCs =
length Bs
        and args-BC:(∀ i<length Bs. Bs!i ⊔ Cs!i = Some (BCs ! i))
        using join-fun-fun[OF Some[unfolded FunB FunC]] by blast
       show ?thesis proof(cases match B (to-pterm (lhs α)))
        case None
        then have ¬ matches BC (to-pterm (lhs α))
          using join-pterm-linear-subst ‹B ⊔ C = Some BC› lin by (metis
match-complete' matches-iff option.simps(3))
        then have A ⊔ BC = None unfolding RuleA BC
         by (smt (verit) join.simps(5) match-matches matches-iff not-Some-eq
option.simps(4))
        then show ?thesis
         unfolding ‹A ⊔ B = None› ‹B ⊔ C = Some BC› by simp
       next
        case (Some σ)
        with None have those (map2 (⊔) As (map σ (var-rule α))) = None
         unfolding RuleA FunB using not-None-eq by fastforce
        then obtain i where i:i < length (var-rule α) map2 (⊔) As (map σ
(var-rule α)) ! i = None
         by (smt (verit, best) RuleA(3) length-map length-zip list-all-length
```

min.idem those-not-none-xs)
let *?x*=*var-rule* α ! *i*
from *i* **have** *none-at-i*:*As* ! *i* $\sqcup$ σ *?x* = *None*
using *RuleA*(*3*) **by** *simp*
show *?thesis* **proof**(*cases match C* (*to-pterm* (*lhs* α)))
case *None*
then have ¬ *matches BC* (*to-pterm* (*lhs* α))
using *join-pterm-linear-subst* ‹*B* $\sqcup$ *C* = *Some BC*› *lin* **by** (*metis match-complete*′ *matches-iff option.simps*(*3*))
then have *A* $\sqcup$ *BC* = *None* **unfolding** *RuleA BC*
by (*smt* (*verit*) *join.simps*(*5*) *match-matches matches-iff not-Some-eq option.simps*(*4*))
then show *?thesis*
unfolding ‹*A* $\sqcup$ *B* = *None*› ‹*B* $\sqcup$ *C* = *Some BC*› **by** *simp*
next
case (*Some* τ)
then obtain ϱ **where** *rho*:(∀ *x*∈*vars-term* (*to-pterm* (*lhs* α)). σ *x* $\sqcup$ τ *x* = *Some* (ϱ *x*))
and *BC-rho*:*BC* = (*to-pterm* (*lhs* α)) · ϱ **and** *match-rho*:*match BC* (*to-pterm* (*lhs* α)) = *Some* ϱ
using *join-pterm-subst-Some match-matches* ‹*match B* (*to-pterm* (*lhs* α)) = *Some* σ› ‹*B* $\sqcup$ *C* = *Some BC*› **by** *blast*
have σ *?x* ∈ *wf-pterm R*
using *i*(*1*) ‹*match B* (*to-pterm* (*lhs* α)) = *Some* σ› *subterm*(*3*) **by** (*metis match-well-def vars-to-pterm*)
moreover have τ *?x* ∈ *wf-pterm R*
using *i*(*1*) *Some subterm*(*4*) **by** (*metis match-well-def vars-to-pterm*)

ultimately have *IH*:*join-opt* (*As* ! *i*) (σ *?x* $\sqcup$ τ *?x*) = *join-opt* (τ *?x*) (*As* ! *i* $\sqcup$ σ *?x*)
using *subterm*(*1*) *i*(*1*) *RuleA*(*3*) **by** (*metis RuleA*(*1*) *RuleA*(*4*) *nth-mem supt.arg*)
then have (*As* ! *i*) $\sqcup$ (ϱ *?x*) = *None*
using *none-at-i rho* **by** (*metis i*(*1*) *join-opt.simps*(*1*) *join-opt.simps*(*2*) *lin linear-term-var-vars-term-list nth-mem set-vars-term-list vars-to-pterm*)
then have (*map2* ($\sqcup$) *As* (*map* ϱ (*var-rule* α))) ! *i* = *None*
using *RuleA*(*3*) *i*(*1*) **by** *auto*
then have *those* (*map2* ($\sqcup$) *As* (*map* ϱ (*var-rule* α))) = *None*
by (*metis* (*no-types*, *opaque-lifting*) *RuleA*(*3*) *i*(*1*) *length-map length-zip min.idem not-Some-eq nth-mem those-not-none-x*)
then have *A* $\sqcup$ *BC* = *None*
using *BC RuleA*(*1*) *match-rho* **by** *force*
then show *?thesis*
unfolding ‹*A* $\sqcup$ *B* = *None*› ‹*B* $\sqcup$ *C* = *Some BC*› **by** *simp*
qed
qed
qed *simp*
next
case (*RuleC* β *Cs*)

```
        from None show ?thesis proof(cases B ⊔ C)
          case (Some BC)
            obtain BCs τ where τ:match B (to-pterm (lhs β)) = Some τ and
BC:BC = Prule β BCs
              and l-BCs:length BCs = length Cs and args-BC:∀ i < length Cs. τ
(var-rule β ! i) ⊔ Cs!i = Some (BCs ! i)
              using join-rule-fun Some[unfolded RuleC FunB] subterm(4)[unfolded
RuleC] FunB(1) by (metis Residual-Join-Deletion.join-sym)
            show ?thesis proof(cases match B (to-pterm (lhs α)))
              case None
             with ‹A ⊔ B = None› Some BC RuleA(1) τ show ?thesis by fastforce
            next
              case (Some σ)
               from None obtain i where i:i < length (var-rule α) map2 (⊔) As
(map σ (var-rule α)) ! i = None
                unfolding FunB RuleA join.simps Some[unfolded FunB] option.case
                  by (smt (verit, ccfv-threshold) length-map length-zip list-all-length
min-less-iff-conj option.case-eq-if option.distinct(1) those-not-none-xs)
              let ?x=var-rule α ! i
              have wf-σ-x:σ ?x ∈ wf-pterm R
               using subterm(3) Some i(1) by (metis match-well-def vars-to-pterm)

            from BC None ‹B ⊔ C = Some BC› RuleA show ?thesis proof(cases
α = β)
              case True
              then have σ = τ
                using Some τ by auto
              have IH:join-opt (As ! i) (τ ?x ⊔ Cs ! i) = join-opt (Cs ! i) (As ! i
⊔ τ ?x)
                using subterm(1) i wf-σ-x args-BC by (metis RuleA(1) RuleA(3)
RuleA(4) RuleC(3) RuleC(4) True ‹σ = τ› nth-mem supt.arg)
              have τ ?x ⊔ Cs ! i = Some (BCs ! i)
                using args-BC i RuleC(3) True by force
              with IH i have (As ! i) ⊔ (BCs ! i) = None
                by (simp add: RuleA(3) ‹σ = τ›)
              then have (map2 (⊔) As BCs) ! i = None
                using l-BCs i by (simp add: RuleA(3) RuleC(3) True)
              then have those (map2 (⊔) As BCs) = None
               using l-BCs i those-not-none-x by (metis RuleA(3) RuleC(3) True
length-map length-zip min.idem nth-mem option.exhaust)
              then have A ⊔ BC = None
                by (simp add: BC RuleA(1))
              then show ?thesis
                unfolding None ‹B ⊔ C = Some BC› by auto
            qed simp
          qed
        qed simp
      qed
    next
```

case (*Some AB*)
obtain σ *ABs* **where** *sigma*:*match B* (*to-pterm* (*lhs* α)) = *Some* σ
and *AB*:*AB* = *Prule* α *ABs* **and** *l-ABs*:*length ABs* = *length As*
and *args-AB*:($\forall$ *i*<*length As*. *As*!*i* $\sqcup$ σ (*var-rule* α ! *i*) = *Some* (*ABs* ! *i*))
using *join-sym join-rule-fun*[*OF Some*[*unfolded FunB RuleA*]] **using** *FunB*(*1*) *RuleA*(*1*) *subterm.prems*(*1*) **by** *blast*
from *subterm*(*4*) *FunB*(*1*) **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*VarC x*)
have *match* (*Var x*) (*to-pterm* (*lhs* α)) = *None*
unfolding *lhs* **using** *match-matches not-Some-eq* **by** *fastforce*
then show *?thesis* **unfolding** *Some* **unfolding** *RuleA FunB AB VarC* **by** *simp*
next
case (*FunC Cs g*)
show *?thesis* **proof**(*cases f* = *g* $\wedge$ *length Bs* = *length Cs*)
case *True*
show *?thesis* **proof**(*cases match C* (*to-pterm* (*lhs* α)))
case *None*
then have $*$:*C* $\sqcup$ *AB* = *None*
unfolding *AB FunC* **by** *simp*
with *Some* **show** *?thesis* **proof**(*cases B* $\sqcup$ *C*)
case (*Some BC*)
with *None* **have** *match BC* (*to-pterm* (*lhs* α)) = *None*
by (*metis* (*no-types*, *lifting*) *domD domIff join-pterm-linear-subst lin match-complete*′ *match-lhs-subst option.simps*(*3*))
moreover obtain *BCs* **where** *BC* = *Pfun f BCs*
by (*metis FunB*(*1*) *FunC*(*1*) *Some join-fun-fun*)
ultimately show *?thesis*
using $*$ **unfolding** ‹*A* $\sqcup$ *B* = *Some AB*› *AB* **unfolding** *RuleA Some* **unfolding** *FunC* **by** (*simp add*: *join-sym*)
qed *simp*
next
case (*Some* τ)
{fix *i* **assume** *i*:*i* < *length As*
have (*map* σ (*var-rule* α)) ! *i* $\in$ *wf-pterm R*
using *i match-well-def*[*OF subterm*(*3*) *sigma*] *RuleA*(*3*) **by** (*simp add*: *vars-to-pterm*)
moreover have (*map* τ (*var-rule* α)) ! *i* $\in$ *wf-pterm R*
using *i match-well-def*[*OF subterm*(*4*) *Some*] *RuleA*(*3*) **by** (*simp add*: *vars-to-pterm*)
ultimately have *join-opt* (*As* ! *i*) ((*map* σ (*var-rule* α) ! *i*) $\sqcup$ (*map* τ (*var-rule* α) ! *i*)) = (*map* τ (*var-rule* α) ! *i*) $\sqcup$ *ABs* ! *i*
using *subterm*(*1*) *RuleA*(*1*,*3*,*4*) *i args-AB True* **by** (*metis* (*no-types*, *lifting*) *join-opt.simps*(*1*) *nth-map nth-mem supt.arg*)
}note *IH*=*this*
show *?thesis* **proof**(*cases B* $\sqcup$ *C*)
case *None*
with *sigma Some* **obtain** *x* **where** *x* $\in$ *vars-term* (*lhs* α) **and** σ *x* $\sqcup$

$\tau\ x = None$
using *join-pterm-subst-None* **by** (*metis lhs-subst-trivial match-lhs-subst option.sel set-vars-term-list vars-to-pterm*)
then obtain i **where** i:$i < length\ (var\text{-}rule\ \alpha)\ (map\ \sigma\ (var\text{-}rule\ \alpha)\ !\ i) \sqcup (map\ \tau\ (var\text{-}rule\ \alpha)\ !\ i) = None$
by (*metis* (*no-types*, *opaque-lifting*) *in-set-idx lin linear-term-var-vars-term-list nth-map set-vars-term-list*)
with *IH* **have** $map2\ (\sqcup)\ (map\ \tau\ (var\text{-}rule\ \alpha))\ ABs\ !\ i = None$
using *RuleA*(*3*) *l-ABs* **by** *fastforce*
with *i*(*1*) **have** $those\ (map2\ (\sqcup)\ (map\ \tau\ (var\text{-}rule\ \alpha))\ ABs) = None$
using *those-not-none-x* **by** (*metis* (*no-types*, *opaque-lifting*) *RuleA*(*3*) *l-ABs length-map length-zip min.idem nth-mem option.exhaust*)
with *Some* **show** *?thesis* **unfolding** $\langle A \sqcup B = Some\ AB\rangle$ *AB None* **unfolding** *FunC* **by** *simp*
next
case (*Some BC*)
from *sigma Some* **obtain** ϱ **where** *rho*:$(\forall x{\in}vars\text{-}term\ (to\text{-}pterm\ (lhs\ \alpha)).\ \sigma\ x \sqcup \tau\ x = Some\ (\varrho\ x))$
and *BC-rho*:$BC = (to\text{-}pterm\ (lhs\ \alpha)) \cdot \varrho$ **and** *match-rho*:*match BC* $(to\text{-}pterm\ (lhs\ \alpha)) = Some\ \varrho$
using *join-pterm-subst-Some match-matches* $\langle match\ C\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \tau\rangle$ **by** *blast*
{fix i **assume** i:$i < length\ As$
with *rho* **have** $(map\ \sigma\ (var\text{-}rule\ \alpha))\ !\ i \sqcup (map\ \tau\ (var\text{-}rule\ \alpha))\ !\ i = Some\ ((map\ \varrho\ (var\text{-}rule\ \alpha))\ !\ i)$
by (*metis* (*no-types*, *lifting*) *RuleA*(*3*) *lin linear-term-var-vars-term-list nth-map nth-mem set-vars-term-list vars-to-pterm*)
with *i IH* **have** $As\ !\ i \sqcup (map\ \varrho\ (var\text{-}rule\ \alpha))\ !\ i = (map\ \tau\ (var\text{-}rule\ \alpha))\ !\ i \sqcup ABs\ !\ i$
by *force*
}
then have $map2\ (\sqcup)\ As\ (map\ \varrho\ (var\text{-}rule\ \alpha)) = map2\ (\sqcup)\ (map\ \tau\ (var\text{-}rule\ \alpha))\ ABs$
by (*simp add*: *RuleA*(*3*) *l-ABs map-equality-iff*)
with *match-rho* $\langle match\ C\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \tau\rangle$
show *?thesis* **unfolding** $\langle A \sqcup B = Some\ AB\rangle$ *AB Some*
unfolding *RuleA BC-rho join-opt.simps to-pterm.simps FunC* **by** (*simp add*: *lhs*)
qed
qed
next
case *False*
then consider (*fg*) $f \neq g$ | (*length*) $length\ Bs \neq length\ Cs$ **by** *fastforce*
then show *?thesis* **proof**(*cases*)
case *fg*
from *sigma* **have** $f' = f$
unfolding *FunB lhs to-pterm.simps* **using** *match-matches* **by** *fastforce*

with *fg* **have** $match\ C\ (to\text{-}pterm\ (lhs\ \alpha)) = None$

unfolding *lhs FunC* **using** *domIff match-matches* **by** *fastforce*
with *fg* **show** *?thesis* **unfolding** ‹*A* ⊔ *B* = *Some AB*› **unfolding** *RuleA FunB FunC AB* **by** *simp*
next
case *length*
from *sigma* **have** *length ts* = *length Bs*
using *FunB*(*1*) *lhs match-matches* **by** *fastforce*
then have *match C* (*to-pterm* (*lhs* α)) = *None*
unfolding *FunC lhs* **using** *length*
by (*smt* (*verit*, *del-insts*) *eval-term.simps*(*2*) *length-map match-matches option.exhaust term.inject*(*2*) *to-pterm.simps*(*2*))
with *length* **show** *?thesis* **unfolding** ‹*A* ⊔ *B* = *Some AB*› **unfolding** *RuleA FunB FunC AB* **by** *simp*
qed
qed
next
case (*RuleC* β *Cs*)
show *?thesis* **proof**(*cases* α = β)
case *True*
{fix *i* **assume** *i*:*i* < *length As*
have (*map* σ (*var-rule* α))!*i* ∈ *wf-pterm R*
using *i match-well-def*[*OF subterm*(*3*) *sigma*] *RuleA*(*3*) **by** (*simp add*: *vars-to-pterm*)
then have *join-opt* (*As*!*i*) ((*map* σ (*var-rule* α) ! *i*) ⊔ *Cs* ! *i*) = *Cs* ! *i* ⊔ *ABs* ! *i*
using *subterm*(*1*) *RuleA*(*1*,*3*,*4*) *RuleC i args-AB True* **by** (*metis* (*no-types*, *lifting*) *join-opt.simps*(*1*) *nth-map nth-mem supt.arg*)
}note *IH*=*this*
show *?thesis* **proof**(*cases those* (*map2* (⊔) (*map* σ (*var-rule* α)) *Cs*))
case *None*
with *sigma* **have** *BC*:*B* ⊔ *C* = *None*
unfolding *FunB RuleC True* **by** *simp*
from *None* **obtain** *i* **where** *i*:*i* < *length* (*map2* (⊔) (*map* σ (*var-rule* α)) *Cs*) **and** (*map2* (⊔) (*map* σ (*var-rule* α)) *Cs*) ! *i* = *None*
using *list-all-length those-not-none-xs* **by** *blast*
with *IH* **have** (*map2* (⊔) *Cs ABs*)!*i* = *None*
using *RuleA*(*3*) *l-ABs RuleC*(*3*) **by** *fastforce*
with *i*(*1*) **have** *those* (*map2* (⊔) *Cs ABs*) = *None*
using *l-ABs RuleA*(*3*) *RuleC*(*3*) *those-not-none-x* **unfolding** *True*
by (*metis length-map length-zip not-Some-eq nth-mem*)
then show *?thesis* **unfolding** *BC* ‹*A* ⊔ *B* = *Some AB*› **unfolding** *AB RuleC True* **by** *simp*
next
case (*Some BCs*)
with *sigma* **have** *BC*:*B* ⊔ *C* = *Some* (*Prule* α *BCs*)
unfolding *FunB RuleC True* **by** *simp*
{fix *i* **assume** *i*:*i* < *length As*
with *Some* **have** (*map* σ (*var-rule* α)) ! *i* ⊔ *Cs* ! *i* = *Some* (*BCs* ! *i*)
using *RuleA*(*3*) *RuleC*(*3*) *True those-some2* **by** *fastforce*

```
            with i IH have As ! i ⊔ BCs ! i = Cs ! i ⊔ ABs ! i
              by force
            }
            moreover have length Cs = length BCs
              using RuleC(3) Some True length-those by fastforce
            ultimately have map2 (⊔) As BCs = map2 (⊔) Cs ABs
              using RuleA(3) l-ABs map-equality-iff
                by (smt (verit, ccfv-threshold) RuleC(3) Some True length-map
length-those length-zip nth-zip old.prod.case)
          then show ?thesis unfolding ‹A ⊔ B = Some AB› BC AB unfolding
RuleA RuleC True by simp
          qed
        next
          case False
          show ?thesis proof(cases B ⊔ C)
            case None
             show ?thesis unfolding None ‹A ⊔ B = Some AB› unfolding AB
RuleC using False by simp
          next
            case (Some BC)
            then obtain BCs where BC = Prule β BCs
               unfolding RuleC FunB by (metis Residual-Join-Deletion.join-sym
RuleC(1) join-rule-fun subterm.prems(3))
             with False show ?thesis unfolding ‹A ⊔ B = Some AB› Some AB
unfolding RuleC RuleA by simp
          qed
        qed
      qed
    next
      case (RuleB β Bs)
      show ?thesis proof(cases A ⊔ B)
        case None
        then show ?thesis proof(cases α = β)
          case True
          then have l-As-Bs:length As = length Bs
            by (simp add: RuleA(3) RuleB(3))
          with None obtain i where i:i < length As As ! i ⊔ Bs ! i = None
             unfolding True RuleA RuleB unfolding join.simps by (smt (verit)
RuleB(3) length-map length-zip list-all-length
            map-nth-eq-conv min-less-iff-conj nth-zip old.prod.case option.case-eq-if
option.distinct(1) those-not-none-xs)
        from subterm(4) show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
          case (VarC x)
          show ?thesis unfolding RuleA RuleB VarC
            by (metis None Residual-Join-Deletion.join-sym RuleA(1) RuleB(1)
RuleB(2) join-opt.simps(2) left-lin-no-var-lhs.join-var-rule left-lin-no-var-lhs-axioms)
        next
```

case (*FunC Cs f*)
from *None* **show** *?thesis* **proof**(*cases B ⊔ C*)
case (*Some BC*)
obtain *BCs τ* **where** *τ*:*match C* (*to-pterm* (*lhs β*)) = *Some τ* **and** *BC*:*BC* = *Prule β BCs*
and *l-BCs*:*length BCs* = *length Bs* **and** *args-BC*:∀ *i* < *length Bs*. *Bs* ! *i* ⊔ *τ* (*var-rule β* ! *i*) = *Some* (*BCs* ! *i*)
using *join-rule-fun Some*[*unfolded RuleB FunC*] *subterm*(*3*)[*unfolded RuleB*] *FunC*(*1*) **by** *metis*
let *?x*=*var-rule β* ! *i*
have *IH*:*join-opt* (*As* ! *i*) (*Bs* ! *i* ⊔ *τ ?x*) = *join-opt* (*τ ?x*) (*As* ! *i* ⊔ *Bs* ! *i*)
by (*metis RuleA*(*1*) *RuleA*(*3*) *RuleA*(*4*) *RuleB*(*3*) *RuleB*(*4*) *True τ i*(*1*) *match-well-def nth-mem subterm.hyps subterm.prems*(*3*) *supt.arg vars-to-pterm*)
with *i* **have** *join-opt* (*As* ! *i*) (*Bs* ! *i* ⊔ *τ ?x*) = *None*
by *simp*
then have *As* ! *i* ⊔ *BCs* ! *i* = *None*
using *args-BC i*(*1*) *l-As-Bs* **by** *auto*
then have (*map2* (⊔) *As BCs*) ! *i* = *None*
using *i*(*1*) *l-As-Bs l-BCs* **by** *force*
then have *those* (*map2* (⊔) *As BCs*) = *None*
by (*metis i*(*1*) *l-As-Bs l-BCs length-map length-zip min.idem not-None-eq nth-mem those-not-none-x*)
then show *?thesis*
using *None RuleA*(*1*) *True BC Some* **by** *auto*
qed *simp*
next
case (*RuleC γ Cs*)
from *None* **show** *?thesis* **proof**(*cases B ⊔ C*)
case (*Some BC*)
then have *β* = *γ*
unfolding *RuleB RuleC* **by** (*metis join.simps*(*3*) *option.distinct*(*1*))
from *Some* **obtain** *BCs* **where** *BC*:*BC* = *Prule β BCs* **and** *l-BCs*:*length BCs* = *length Bs* **and**
args-BC:∀ *i* < *length Bs*. *Bs* ! *i* ⊔ *Cs* ! *i* = *Some* (*BCs* ! *i*)
using *RuleB*(*1*) *RuleC*(*1*) *join-rule-rule subterm.prems*(*2*) *subterm.prems*(*3*) **by** *blast*
have *IH*:*join-opt* (*As* ! *i*) (*Bs* ! *i* ⊔ *Cs* ! *i*) = *join-opt* (*Cs* ! *i*) (*As* ! *i* ⊔ *Bs* ! *i*)
using *subterm*(*1*) **by** (*metis RuleA*(*1*) *RuleA*(*3*) *RuleA*(*4*) *RuleB*(*4*) *RuleC*(*3*) *RuleC*(*4*) *True* ‹*β* = *γ*› *i*(*1*) *l-As-Bs nth-mem supt.arg*)
then have *As* ! *i* ⊔ *BCs* ! *i* = *None*
using *args-BC i*(*1*) *l-As-Bs i*(*2*) **by** *fastforce*
then have (*map2* (⊔) *As BCs*) ! *i* = *None*
using *i*(*1*) *l-As-Bs l-BCs* **by** *force*
then have *those* (*map2* (⊔) *As BCs*) = *None*
by (*metis i*(*1*) *l-As-Bs l-BCs length-map length-zip min.idem not-None-eq nth-mem those-not-none-x*)
then show *?thesis*

```
            using None RuleA(1) True BC Some by auto
          qed simp
        qed
      next
        case False
        then show ?thesis proof(cases C)
          case (Var x)
          show ?thesis unfolding RuleA RuleB Var
            by (metis None Residual-Join-Deletion.join-sym RuleA(1) RuleB(1)
RuleB(2) join-opt.simps(2) join-var-rule)
        next
          case (Pfun f Cs)
          show ?thesis unfolding RuleA RuleB Pfun
          by (metis (no-types, lifting) False RuleB(1) join.simps(3) join-opt.elims
join-opt.simps(2) join-rule-fun subterm.prems(2))
        next
          case (Prule γ Cs)
          show ?thesis unfolding RuleA RuleB Prule
                by (smt (verit, ccfv-threshold) False Prule RuleA(1) RuleB(1)
join-opt.elims join-rule-rule join-wf-pterm subterm.prems(1) subterm.prems(2) sub-
term.prems(3))
        qed
      qed
    next
      case (Some AB)
      then have alpha-beta:β = α
        unfolding RuleA RuleB by (metis join.simps(3) option.distinct(1))
      with Some obtain ABs where AB:AB = Prule α ABs and l-AB-A:length
ABs = length As
        and args-AB:(∀ i<length Bs. As!i ⊔ Bs!i = Some (ABs ! i))
      by (smt (verit, ccfv-SIG) RuleA(1) RuleB(1) join-rule-rule subterm.prems(1)
subterm.prems(2))
    from subterm(4) show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
        case (VarC x)
        have match (Var x) (to-pterm (Fun f' ts)) = None
       by (metis case-optionE match-matches option.disc-eq-case(2) subst-apply-eq-Var
term.distinct(1) to-pterm.simps(2))
       then show ?thesis unfolding Some AB unfolding RuleB VarC join.simps
alpha-beta by (simp add: lhs)
      next
        case (FunC Cs f)
        show ?thesis proof(cases match C (to-pterm (lhs α)))
          case None
           then show ?thesis unfolding Some AB unfolding RuleB alpha-beta
FunC by simp
        next
          case (Some σ)
          {fix i assume i:i < length As
```

```
          have (map σ (var-rule α))!i ∈ wf-pterm R
            using i match-well-def[OF subterm(4) Some] RuleA(3) by (simp
add: vars-to-pterm)
           with i have join-opt (As ! i) (Bs ! i ⊔ (map σ (var-rule α) ! i)) =
map σ (var-rule α) ! i ⊔ ABs ! i
             using subterm(1) RuleA RuleB by (metis alpha-beta args-AB
join-opt.simps(1) nth-mem supt.arg)
         }note IH=this
         show ?thesis proof(cases those (map2 (⊔) Bs (map σ (var-rule α))))
          case None
         from None obtain i where i:i < length (map2 (⊔) Bs (map σ (var-rule
α))) and (map2 (⊔) Bs (map σ (var-rule α))) ! i = None
           using list-all-length those-not-none-xs by blast
          with IH have (map2 (⊔) (map σ (var-rule α)) ABs)!i = None
           using RuleA(3) l-AB-A by fastforce
         with i(1) have those (map2 (⊔) (map σ (var-rule α)) ABs) = None
         using l-AB-A RuleA(3) those-not-none-x by (metis RuleB(3) alpha-beta
length-map length-zip nth-mem option.exhaust)
          with ‹A ⊔ B = Some AB› Some None show ?thesis unfolding AB
RuleB FunC alpha-beta by fastforce
         next
          case (Some BCs)
          then have BC:B ⊔ C = Some (Prule α BCs)
            unfolding RuleB FunC alpha-beta using ‹match C (to-pterm (lhs
α)) = Some σ› by (simp add: FunC(1))
          then have l-BCs:length BCs = length As
           using RuleA(3) RuleB(3) Some alpha-beta length-those by force
          {fix i assume i < length As
           then have As!i ⊔ BCs!i = (map σ (var-rule α)) ! i ⊔ ABs ! i
            using IH Some l-BCs length-those those-some2 by fastforce
          }
          then have map2 (⊔) As BCs = map2 (⊔) (map σ (var-rule α)) ABs
           by (simp add: RuleA(3) l-AB-A l-BCs map-equality-iff)
          then show ?thesis using ‹match C (to-pterm (lhs α)) = Some σ›
            unfolding ‹A ⊔ B = Some AB› AB BC unfolding RuleA FunC
join-opt.simps join.simps by simp
         qed
        qed
       next
        case (RuleC γ Cs)
        show ?thesis proof(cases α = γ)
         case True
         {fix i assume i:i < length As
          then have join-opt (As ! i) (Bs ! i ⊔ Cs ! i) = Cs ! i ⊔ ABs ! i
           using subterm(1) RuleA RuleB RuleC True alpha-beta args-AB by
(metis join-opt.simps(1) nth-mem supt.arg)
         }note IH=this
         show ?thesis proof(cases those (map2 (⊔) Bs Cs))
          case None
```

```
            then obtain i where i:i < length (map2 (⊔) Bs Cs) (map2 (⊔) Bs
Cs)!i = None
              using list-all-length those-not-none-xs by blast
            with IH have map2 (⊔) Cs ABs ! i = None
              using RuleA(3) RuleC(3) True l-AB-A by fastforce
            with i(1) have those (map2 (⊔) Cs ABs) = None
                by (metis RuleA(3) RuleB(3) RuleC(3) True alpha-beta l-AB-A
length-map length-zip not-Some-eq nth-mem those-not-none-x)
          with None show ?thesis unfolding Some AB unfolding RuleC RuleB
True alpha-beta by simp
        next
          case (Some BCs)
          then have BC:B ⊔ C = Some (Prule α BCs)
            by (simp add: RuleB(1) RuleC(1) True alpha-beta)
          {fix i assume i < length As
            with IH have As!i ⊔ BCs!i = Cs!i ⊔ ABs!i
          using RuleA(3) RuleB(3) RuleC(3) Some True alpha-beta those-some2
by fastforce
          }
          moreover have length BCs = length As
          using RuleA(3) RuleB(3) RuleC(3) Some True alpha-beta length-those
by force
            ultimately have those (map2 (⊔) As BCs) = those (map2 (⊔) Cs
ABs)
              by (smt (verit, ccfv-SIG) RuleA(3) RuleB(3) RuleC(3) Some
True alpha-beta l-AB-A length-map length-those length-zip map-equality-iff nth-zip
old.prod.case)
          then show ?thesis unfolding ‹A ⊔ B = Some AB› AB BC unfolding
RuleA RuleC True alpha-beta by simp
        qed
      next
        case False
          then show ?thesis unfolding ‹A ⊔ B = Some AB› AB unfolding
RuleA RuleB RuleC
          by (simp add: alpha-beta)
      qed
    qed
  qed
 qed
qed
```

Preparation for well-definedness result for $\bigsqcup$.

```
lemma join-triple-defined:
  assumes A ∈ wf-pterm R B ∈ wf-pterm R C ∈ wf-pterm R
    and A ⊔ B ≠ None B ⊔ C ≠ None A ⊔ C ≠ None
  shows join-opt A (B ⊔ C) ≠ None
  using assms proof(induct A arbitrary:B C rule:subterm-induct)
  case (subterm A)
```

from *subterm*(*5*) **obtain** *AB* **where** *joinAB*:*A* ⊔ *B* = *Some AB* **by** *blast*
from *subterm*(*6*) **obtain** *BC* **where** *joinBC*:*B* ⊔ *C* = *Some BC* **by** *blast*
from *subterm*(*7*) **obtain** *AC* **where** *joinAC*:*A* ⊔ *C* = *Some AC* **by** *blast*
from *subterm*(*2*) **show** *?case* **proof**(*cases A rule*:*wf-pterm.cases*[*case-names VarA FunA RuleA*])
case (*VarA x*)
from *subterm*(*3*,*5*) **show** *?thesis* **proof**(*cases B rule*:*wf-pterm.cases*[*case-names VarB FunB RuleB*])
case (*VarB y*)
from *subterm*(*5*) **have** *x*:*x* = *y* **unfolding** *VarA VarB* **by** (*meson join.simps*(*1*))

from *subterm*(*4*,*6*) **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*VarC z*)
from *subterm*(*6*) **show** *?thesis* **unfolding** *VarA VarB x VarC*
by (*metis join.simps*(*1*) *join-opt.simps*(*1*))
next
case (*RuleC α Cs*)
from *subterm*(*5*−) **show** *?thesis* **unfolding** *VarA VarB RuleC x*
by (*metis Residual-Join-Deletion.join-sym RuleC*(*1*) *VarA join-opt.elims join-with-source option.sel source.simps*(*1*) *source-join subterm.prems*(*1*) *subterm.prems*(*3*) *to-pterm.simps*(*1*) *x*)
qed (*simp add*: *VarB*)
next
case (*RuleB α Bs*)
from *subterm*(*2*−) *VarA no-var-lhs RuleB* **show** *?thesis*
by (*metis join-sym join-opt.elims join-wf-pterm join-with-source source.simps*(*1*) *source-join to-pterm.simps*(*1*))
qed (*simp add*: *VarA*)
next
case (*FunA As f*)
from *subterm*(*3*,*5*) **show** *?thesis* **proof**(*cases B rule*:*wf-pterm.cases*[*case-names VarB FunB RuleB*])
case (*FunB Bs g*)
from *subterm*(*5*) **have** *fg*:*f* = *g* **and** *l-A-B*:*length As* = *length Bs*
unfolding *FunA FunB* **by** (*meson join.simps*(*2*))+
obtain *ABs* **where** *AB*:*AB* = *Pfun f ABs* **and** *l-AB-A*:*length ABs* = *length As*
and *args-AB*:(∀ *i*<*length Bs*. *As*!*i* ⊔ *Bs*!*i* = *Some* (*ABs* ! *i*))
using *join-fun-fun*[*OF joinAB*[*unfolded FunA FunB*]] **by** *fastforce*
from *subterm*(*4*,*6*) **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*FunC Cs h*)
from *subterm*(*6*) **have** *gh*:*g* = *h* **and** *l-B-C*:*length Bs* = *length Cs*
unfolding *FunB FunC* **by** (*meson join.simps*(*2*))+
from *subterm*(*7*) **have** *fh*:*f* = *h* **and** *l-A-C*:*length As* = *length Cs*
unfolding *FunA FunC* **by** (*meson join.simps*(*2*))+
obtain *BCs* **where** *BC*:*BC* = *Pfun g BCs* **and** *l-BC-B*:*length BCs* = *length Bs*

and *args-BC*:(∀ *i*<*length BCs*. *Bs*!*i* ⊔ *Cs*!*i* = *Some* (*BCs* ! *i*))
using *join-fun-fun*[*OF joinBC*[*unfolded FunB FunC*]] **by** *fastforce*
obtain *ACs* **where** *AC*:*AC* = *Pfun h ACs* **and** *l-AC-C*:*length ACs* = *length Cs*
and *args-AC*:(∀ *i*<*length ACs*. *As*!*i* ⊔ *Cs*!*i* = *Some* (*ACs* ! *i*))
using *join-fun-fun*[*OF joinAC*[*unfolded FunA FunC*]] **by** *fastforce*
have *those* (*map2* (⊔) *As BCs*) ≠ *None* **proof**−
{**fix** *i* **assume** *i*:*i* < *length* (*zip As BCs*)
from *FunA FunB FunC i* **have** *join-opt* (*As*!*i*) ((*Bs*!*i*) ⊔ (*Cs*!*i*)) ≠ *None*
using *subterm*(*1*) *l-A-B l-B-C l-AC-C* **by** (*smt* (*verit*, *ccfv-threshold*) *args-AB args-AC args-BC length-zip min-less-iff-conj nth-mem option.distinct*(*1*) *supt.arg*)
then have (*map2* (⊔) *As BCs*)!*i* ≠ *None*
using *i args-BC* **by** *simp*
}
then show *?thesis*
by (*simp add*: *list-all-length those-not-none-xs*)
qed
then show *?thesis*
unfolding *joinBC BC* **unfolding** *FunA fg gh join-opt.simps*
by (*simp add*: *l-A-B l-BC-B option.case-eq-if*)
next
case (*RuleC α Cs*)
from *joinBC subterm*(*4*) **obtain** *σ BCs* **where** *match-lhs-B*:*match B* (*to-pterm* (*lhs α*)) = *Some σ*
and *BC*:*BC* = *Prule α BCs* **and** *l-BC-C*:*length BCs* = *length Cs*
and *args-BC*:(∀ *i*<*length Cs*. *Cs* ! *i* ⊔ *σ* (*var-rule α* ! *i*) = *Some* (*BCs* ! *i*))
unfolding *FunB RuleC* **using** *join-rule-fun RuleC*(*1*,*2*,*3*) *join-sym* **by** *metis*
from *joinAC subterm*(*4*) **obtain** *τ ACs* **where** *match-lhs-A*:*match A* (*to-pterm* (*lhs α*)) = *Some τ*
and *AC*:*AC* = *Prule α ACs* **and** *l-AC-C*:*length ACs* = *length Cs*
and *args-AC*:(∀ *i*<*length Cs*. *Cs* ! *i* ⊔ *τ* (*var-rule α* ! *i*) = *Some* (*ACs* ! *i*))
unfolding *FunA RuleC* **using** *join-rule-fun RuleC*(*3*) *join-sym* **by** *metis*
have *those* (*map2* (⊔) (*map τ* (*var-rule α*)) *BCs*) ≠ *None* **proof**−
{**fix** *i* **assume** *i*:*i* < *length* (*zip* (*map τ* (*var-rule α*)) *BCs*)
from *i* **obtain** *x* **where** *x*:*var-rule α* ! *i* = *x x* ∈ *vars-term* (*to-pterm* (*lhs α*))
by (*metis* (*no-types*, *lifting*) *comp-apply length-map length-zip min-less-iff-conj nth-mem set-remdups set-rev set-vars-term-list vars-to-pterm*)
have *τ* (*var-rule α* ! *i*) ◁ *A* **proof**−
from *RuleC*(*2*) *no-var-lhs* **obtain** *f' ts* **where** *lhs α* = *Fun f' ts* **by** *fastforce*
with *x* **show** *?thesis*
using *subst-image-subterm*[*of x*] *match-lhs-A* **unfolding** *FunA*
by (*smt* (*verit*) *match-matches to-pterm.simps*(*2*))
qed
moreover have *τ* (*var-rule α* ! *i*) ∈ *wf-pterm R*
using *i match-well-def*[*OF subterm*(*2*) *match-lhs-A*] **by** (*simp add*:

vars-to-pterm)
moreover have σ (*var-rule* α ! *i*) $\in$ *wf-pterm R*
using *i match-well-def*[*OF subterm*(*3*) *match-lhs-B*] **by** (*simp add*: *vars-to-pterm*)
moreover have τ (*var-rule* α ! *i*) $\sqcup$ σ (*var-rule* α ! *i*) $\neq$ *None*
using *join-pterm-subst-Some x match-lhs-B match-lhs-A match-matches subterm.prems*(*4*) *x* **by** *blast*
moreover have τ (*var-rule* α ! *i*) $\sqcup$ (*Cs*!*i*) $\neq$ *None*
using *args-AC* **by** (*metis join-sym RuleC*(*3*) *i length-map length-zip min-less-iff-conj option.distinct*(*1*))
moreover have σ (*var-rule* α ! *i*) $\sqcup$ (*Cs*!*i*) $\neq$ *None*
using *args-BC* **by** (*metis join-sym RuleC*(*3*) *i length-map length-zip min-less-iff-conj option.distinct*(*1*))
ultimately have *IH*:*join-opt* (τ (*var-rule* α ! *i*)) (σ (*var-rule* α ! *i*) $\sqcup$ (*Cs*!*i*)) $\neq$ *None*
using *RuleC*(*3*,*4*) *subterm*(*1*) *i* **by** *simp*
from *IH* **have** (τ (*var-rule* α ! *i*)) $\sqcup$ (*BCs*!*i*) $\neq$ *None*
using *i args-BC l-BC-C join-sym* **by** (*metis* (*no-types*, *opaque-lifting*) *join-opt.simps*(*1*) *length-zip min-less-iff-conj*)
then have (*map2* ($\sqcup$) (*map* τ (*var-rule* α)) *BCs*)!*i* $\neq$ *None*
unfolding *nth-map*[*OF i*] **using** *i* **by** *auto*
}
then show *?thesis* **by** (*simp add*: *list-all-length those-not-none-xs*)
qed
with *match-lhs-A* **show** *?thesis*
unfolding *joinBC BC FunA* **unfolding** *fg join-opt.simps join.simps*(*7*) **by** *force*
qed (*simp add*:*FunB*)
next
case (*RuleB* α *Bs*)
from *joinAB* **have** $*$:*Prule* α *Bs* $\sqcup$ *Pfun f As* = *Some AB* **unfolding** *FunA RuleB* **using** *join-sym* **by** *metis*
obtain σ *ABs* **where** *match-lhs-A*:*match A* (*to-pterm* (*lhs* α)) = *Some* σ
and *AB*:*AB* = *Prule* α *ABs* **and** *l-A-AB*:*length ABs* = *length Bs*
and *args-AB*:($\forall$ *i*<*length Bs*. *Bs* ! *i* $\sqcup$ σ (*var-rule* α ! *i*) = *Some* (*ABs* ! *i*))
unfolding *FunA RuleB* **using** *join-rule-fun*[*OF* $*$ *subterm*(*3*)[*unfolded FunA RuleB*]] *RuleB*(*3*) **by** *fastforce*
from *subterm*(*4*,*7*) **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*FunC Cs g*)
from *joinBC* **have** $*$:*Prule* α *Bs* $\sqcup$ *Pfun g Cs* = *Some BC* **unfolding** *FunC RuleB* **by** *metis*
from *subterm*(*3*) **obtain** τ *BCs* **where** *match-lhs-C*:*match C* (*to-pterm* (*lhs* α)) = *Some* τ
and *BC*:*BC* = *Prule* α *BCs* **and** *l-BC-B*:*length BCs* = *length Bs*
and *args-BC*:($\forall$ *i*<*length Bs*. *Bs* ! *i* $\sqcup$ τ (*var-rule* α ! *i*) = *Some* (*BCs* ! *i*))
unfolding *FunC RuleB* **using** *join-rule-fun*[*OF joinBC*[*unfolded FunC RuleB*]] *RuleB*(*3*) **by** *fastforce*
have *those* (*map2* ($\sqcup$) (*map* σ (*var-rule* α)) *BCs*) $\neq$ *None* **proof**−

{**fix** i **assume** i:$i < length\ (zip\ (map\ \tau\ (var\text{-}rule\ \alpha))\ BCs)$
from i **obtain** x **where** x:$var\text{-}rule\ \alpha\ !\ i = x\ x \in vars\text{-}term\ (to\text{-}pterm\ (lhs\ \alpha))$
by (*metis* (*no-types*, *lifting*) *comp-apply length-map length-zip min-less-iff-conj nth-mem set-remdups set-rev set-vars-term-list vars-to-pterm*)
have $\sigma\ (var\text{-}rule\ \alpha\ !\ i) \lhd A$ **proof**−
from *RuleB*(*2*) *no-var-lhs* **obtain** $f'\ ts$ **where** $lhs\ \alpha = Fun\ f'\ ts$ **by** *fastforce*
with x **show** *?thesis*
using *subst-image-subterm*[*of* x] *match-lhs-A* **unfolding** *FunA*
by (*smt* (*verit*) *match-matches to-pterm.simps*(*2*))
qed
moreover have $\sigma\ (var\text{-}rule\ \alpha\ !\ i) \in wf\text{-}pterm\ R$
using *i match-well-def*[*OF subterm*(*2*) *match-lhs-A*] **by** (*simp add*: *vars-to-pterm*)
moreover have $\tau\ (var\text{-}rule\ \alpha\ !\ i) \in wf\text{-}pterm\ R$
using *i match-well-def*[*OF subterm*(*4*) *match-lhs-C*] **by** (*simp add*: *vars-to-pterm*)
moreover have $\sigma\ (var\text{-}rule\ \alpha\ !\ i) \sqcup \tau\ (var\text{-}rule\ \alpha\ !\ i) \neq None$
using *join-pterm-subst-Some x match-lhs-C match-lhs-A match-matches subterm.prems*(*6*) x **by** *blast*
moreover have $(Bs!i) \sqcup \tau\ (var\text{-}rule\ \alpha\ !\ i) \neq None$
using *args-BC i* **by** (*metis RuleB*(*3*) *i length-map length-zip min-less-iff-conj option.distinct*(*1*))
moreover have $\sigma\ (var\text{-}rule\ \alpha\ !\ i) \sqcup (Bs!i) \neq None$
using *args-AB* **by** (*metis join-sym RuleB*(*3*) *i length-map length-zip min-less-iff-conj option.distinct*(*1*))
moreover have $\sigma\ (var\text{-}rule\ \alpha\ !\ i) \sqcup \tau\ (var\text{-}rule\ \alpha\ !\ i) \neq None$
using *join-pterm-subst-Some x match-lhs-C match-lhs-A match-matches subterm.prems*(*6*) x **by** *blast*
ultimately have *IH*:*join-opt* $(\sigma\ (var\text{-}rule\ \alpha\ !\ i))\ ((Bs!i) \sqcup \tau\ (var\text{-}rule\ \alpha\ !\ i)) \neq None$
using *RuleB*(*3*,*4*) *subterm*(*1*) i **by** *simp*
then have $(map2\ (\sqcup)\ (map\ \sigma\ (var\text{-}rule\ \alpha))\ BCs)!i \neq None$
using *i args-BC l-BC-B* **unfolding** *nth-map*[*OF i*] **using** i **by** *auto*
}
then show *?thesis* **by** (*simp add*: *list-all-length those-not-none-xs*)
qed
with *match-lhs-A* **show** *?thesis*
unfolding *joinBC BC FunA* **unfolding** *join-opt.simps join.simps*(*7*) **by** *force*
next
case (*RuleC* β *Cs*)
obtain *BCs* **where** α:$\alpha = \beta$ **and** *l-B-C*:*length Bs* = *length Cs*
and *BC*:*BC* = *Prule* α *BCs* **and** *l-BC-B*:*length BCs* = *length Bs* **and** *args-BC*:$(\forall\ i$<*length Bs*. *Bs* ! $i \sqcup Cs\ !\ i = Some\ (BCs\ !\ i))$
using *join-rule-rule joinBC subterm*(*3*,*4*) **unfolding** *RuleB*(*1*) *RuleC*(*1*) **by** *blast*
from *joinAC match-lhs-A* **have** *args-AC*:$\forall\ i$<*length Cs*. *Cs* ! $i \sqcup \sigma$ (*var-rule*

```
α ! i) ≠ None
      using join-rule-fun by (metis (no-types, lifting) FunA(1) Residual-Join-Deletion.join-sym
RuleC(1) α option.distinct(1) option.inject subterm.prems(3))
       have those (map2 (⊔) (map σ (var-rule α)) BCs) ≠ None proof−
        {fix i assume i:i < length (zip (map σ (var-rule α)) BCs)
          from i obtain x where x:var-rule α ! i = x x ∈ vars-term (to-pterm
(lhs α))
                  by (metis (no-types, lifting) comp-apply length-map length-zip
min-less-iff-conj nth-mem set-remdups set-rev set-vars-term-list vars-to-pterm)
         have σ (var-rule α ! i) ◁ A proof−
           from RuleB(2) no-var-lhs obtain f' ts where lhs α = Fun f' ts by
fastforce
          with x show ?thesis
           using subst-image-subterm[of x] match-lhs-A unfolding FunA
           by (smt (verit) match-matches to-pterm.simps(2))
         qed
         moreover have σ (var-rule α ! i) ∈ wf-pterm R
             using i match-well-def[OF subterm(2) match-lhs-A] by (simp add:
vars-to-pterm)
         moreover have σ (var-rule α ! i) ⊔ (Bs!i) ≠ None
            using args-AB by (metis join-sym RuleB(3) i length-map length-zip
min-less-iff-conj option.distinct(1))
         moreover have (Bs!i) ⊔ (Cs!i) ≠ None
          using args-BC i by (simp add: l-BC-B)
         moreover have σ (var-rule α ! i) ⊔ (Cs!i) ≠ None
          using args-AC by (metis join-sym RuleC(3) α i length-map length-zip
min-less-iff-conj)
          ultimately have IH:join-opt (σ (var-rule α ! i)) ((Bs!i) ⊔ (Cs!i)) ≠
None
           using RuleB(3,4) RuleC(3,4) subterm(1) i l-B-C by auto
         then have (map2 (⊔) (map σ (var-rule α)) BCs)!i ≠ None
          using i args-BC l-BC-B unfolding nth-map[OF i] using i by auto
        }
        then show ?thesis by (simp add: list-all-length those-not-none-xs)
       qed
       with match-lhs-A show ?thesis
        unfolding joinBC BC FunA unfolding join-opt.simps join.simps(7) by
force
     qed (simp add: FunA)
   qed (simp add: FunA)
  next
   case (RuleA α As)
  from subterm(3,5) show ?thesis proof(cases B rule:wf-pterm.cases[case-names
VarB FunB RuleB])
     case (VarB x)
     from subterm(2−) show ?thesis
    by (metis join-sym VarB joinBC join-opt.simps(1) join-with-source source.simps(1)
source-join to-pterm.simps(1))
   next
```

case (*FunB Bs f*)
from *subterm*(*2*) **obtain** σ *ABs* **where** *match-lhs-B*:*match B* (*to-pterm* (*lhs* α)) = *Some* σ
and *AB*:*AB* = *Prule* α *ABs* **and** *l-A-AB*:*length ABs* = *length As*
and *args-AB*:($\forall$ *i*<*length As*. *As* ! *i* $\sqcup$ σ (*var-rule* α ! *i*) = *Some* (*ABs* ! *i*))
unfolding *RuleA FunB* **using** *join-rule-fun*[*OF joinAB*[*unfolded RuleA FunB*]] *RuleA*(*1*,*3*) **by** *fastforce*
from *subterm*(*4*,*6*) **show** *?thesis* **proof**(*cases C rule*:*wf-pterm.cases*[*case-names VarC FunC RuleC*])
case (*FunC Cs g*)
from *subterm*(*2*) **obtain** τ *ACs* **where** *match-lhs-C*:*match C* (*to-pterm* (*lhs* α)) = *Some* τ
and *AC*:*AC* = *Prule* α *ACs* **and** *l-A-AC*:*length ACs* = *length As*
and *args-AC*:($\forall$ *i*<*length As*. *As* ! *i* $\sqcup$ τ (*var-rule* α ! *i*) = *Some* (*ACs* ! *i*))
unfolding *RuleA FunC* **using** *join-rule-fun*[*OF joinAC*[*unfolded RuleA FunC*]] *RuleA*(*1*,*3*) **by** *fastforce*
from *joinBC* **obtain** ϱ **where** $\forall$ *x*$\in$*vars-term* (*to-pterm* (*lhs* α)). σ *x* $\sqcup$ τ *x* = *Some* (ϱ *x*) **and** *BC* = *to-pterm* (*lhs* α) $\cdot$ ϱ
using *join-pterm-subst-Some*[*of to-pterm* (*lhs* α)] *match-lhs-C match-lhs-B* **by** (*smt* (*verit*) *match-matches*)
then obtain *BCs* **where** *args-BC*:($\forall$ *i*<*length As*. σ (*var-rule* α ! *i*) $\sqcup$ τ (*var-rule* α ! *i*) = *Some* (*BCs* ! *i*))
and *BC*:*BC* = (*to-pterm* (*lhs* α)) $\cdot$ $\langle$*BCs*$\rangle_\alpha$ **and** *l-A-BC*:*length As* = *length BCs*
using *subst-imp-mk-subst*[*of BC to-pterm* (*lhs* α)] *RuleA*(*3*)
by (*smt* (*verit*, *del-insts*) *comp-apply nth-mem set-remdups set-rev set-vars-term-list vars-to-pterm*)
from *RuleA*(*2*) *no-var-lhs* **obtain** *f' ts* **where** *lhs*:*lhs* α = *Fun f' ts* **by** *fastforce*
{fix *i* **assume** *i*:*i* < *length As*
from *i* **obtain** *x* **where** *x*:*var-rule* α ! *i* = *x x* $\in$ *vars-term* (*to-pterm* (*lhs* α))
by (*metis RuleA*(*3*) *comp-apply nth-mem set-remdups set-rev set-vars-term-list vars-to-pterm*)
have σ (*var-rule* α ! *i*) $\in$ *wf-pterm R*
using *RuleA*(*3*) *i match-well-def*[*OF subterm*(*3*) *match-lhs-B*] **by** (*simp add*: *vars-to-pterm*)
moreover have τ (*var-rule* α ! *i*) $\in$ *wf-pterm R*
using *RuleA*(*3*) *i match-well-def*[*OF subterm*(*4*) *match-lhs-C*] **by** (*simp add*: *vars-to-pterm*)
moreover have *As*!*i* $\sqcup$ σ (*var-rule* α ! *i*) $\neq$ *None*
using *args-AB i* **by** *auto*
moreover have *As*!*i* $\sqcup$ τ (*var-rule* α ! *i*) $\neq$ *None*
using *args-AC i* **by** *auto*
moreover have σ (*var-rule* α ! *i*) $\sqcup$ τ (*var-rule* α ! *i*) $\neq$ *None*
using *args-BC i* **by** *auto*
ultimately have *IH*:*join-opt* (*As*!*i*) (σ (*var-rule* α ! *i*) $\sqcup$ τ (*var-rule* α ! *i*)) $\neq$ *None*
using *RuleA*(*3*,*4*) *subterm*(*1*) *i* **by** (*metis RuleA*(*1*) *nth-mem supt.arg*)

```
        then have As!i ⊔ BCs!i ≠ None
          using i args-BC by auto
        }
        with subterm(2) show ?thesis
       unfolding joinBC BC RuleA(1) unfolding join-opt.simps using join-rule-lhs
l-A-BC by auto
      next
        case (RuleC β Cs)
          from joinBC subterm(4) obtain τ BCs where match-lhs-B2:match B
(to-pterm (lhs β)) = Some τ
         and BC:BC = Prule β BCs and l-BC-C:length BCs = length Cs
        and args-BC:(∀ i<length Cs. Cs ! i ⊔ τ (var-rule β ! i) = Some (BCs ! i))
          unfolding FunB RuleC using join-rule-fun RuleC(1,2,3) join-sym by
metis
        from joinAC have α:α = β and l-A-C:length As = length Cs
          unfolding RuleA RuleC by(metis join.simps(3) option.distinct(1))+
        have those (map2 (⊔) As BCs) ≠ None proof−
          {fix i assume i:i < length (zip As BCs)
            moreover have τ (var-rule β ! i) ∈ wf-pterm R
              using i match-well-def[OF subterm(3) match-lhs-B2] by (simp add:
RuleA(3) α vars-to-pterm)
            moreover have As!i ⊔ τ (var-rule β ! i) ≠ None
                using join-pterm-subst-Some match-lhs-B subterm(5) α args-AB i
match-lhs-B2 by auto
            moreover have (As!i) ⊔ (Cs!i) ≠ None
              using joinAC RuleA(1) RuleC(1) i join-rule-rule subterm.prems(1)
subterm.prems(3) by fastforce
            moreover have τ (var-rule β ! i) ⊔ (Cs!i) ≠ None
              using args-BC i by (simp add: join-sym l-A-C)
           ultimately have IH:join-opt (As!i) (τ (var-rule β ! i) ⊔ (Cs!i)) ≠ None

              using RuleA(1,3,4) RuleC(3,4) subterm(1) i α by simp
            from IH have (As!i) ⊔ (BCs!i) ≠ None
              using i args-BC by (simp add: join-sym l-BC-C)
            then have (map2 (⊔) As BCs)!i ≠ None
              unfolding nth-map[OF i] using i by auto
          }
          then show ?thesis by (simp add: list-all-length those-not-none-xs)
        qed
        then show ?thesis
          unfolding joinBC BC RuleA α unfolding join-opt.simps join.simps(7)
by force
      qed (simp add:FunB)
    next
      case (RuleB β Bs)
      from joinAB have αβ:α = β and l-A-B:length As = length Bs
        unfolding RuleA RuleB by(metis join.simps(3) option.distinct(1))+
     obtain ABs where AB:AB = Prule α ABs and l-AB-B:length ABs = length
Bs
```

```
      and args-AB:(∀ i<length ABs. As!i ⊔ Bs!i = Some (ABs ! i))
        using join-rule-rule[OF joinAB[unfolded RuleA RuleB]] subterm(2,3)
RuleA(1) RuleB(1) by metis
    from subterm(4,6) show ?thesis proof(cases C rule:wf-pterm.cases[case-names
VarC FunC RuleC])
      case (VarC x)
      from joinBC RuleB(2) no-var-lhs show ?thesis unfolding VarC RuleB
      by (metis Residual-Join-Deletion.join-sym RuleB(1) VarC join-opt.simps(1)
join-with-source source.simps(1) source-join subterm.prems(2) subterm.prems(3)
subterm.prems(4) to-pterm.simps(1))
    next
      case (FunC Cs f)
       from subterm(3) obtain σ BCs where match-lhs-C:match C (to-pterm
(lhs β)) = Some σ
        and BC:BC = Prule β BCs and l-BC-C:length BCs = length Bs
       and args-BC:(∀ i<length Bs. Bs ! i ⊔ σ (var-rule β ! i) = Some (BCs ! i))
        unfolding RuleB using join-rule-fun[OF joinBC[unfolded RuleB FunC]]
RuleB(1,2,3) by (metis FunC(1))
      have those (map2 (⊔) As BCs) ≠ None proof−
       {fix i assume i:i < length (zip As BCs)
        have σ (var-rule β ! i) ∈ wf-pterm R
          using i match-well-def[OF subterm(4) match-lhs-C] by (simp add:
RuleA(3) αβ vars-to-pterm)
        moreover have As!i ⊔ σ (var-rule β ! i) ≠ None
          using match-lhs-C joinAC αβ args-AB i unfolding RuleA(1) FunC
            by (metis (no-types, lifting) RuleA(1) join-rule-fun length-zip
min-less-iff-conj option.distinct(1) option.sel subterm.prems(1))
         ultimately have IH:join-opt (As!i) ((Bs!i) ⊔ (σ (var-rule β ! i))) ≠
None
          using RuleA(1,3,4) subterm(1) i args-AB args-BC
            by (metis (no-types, lifting) RuleB(4) l-AB-B l-A-B length-zip
min-less-iff-conj nth-mem option.distinct(1) supt.arg)
        from IH have (As!i) ⊔ (BCs!i) ≠ None
          using i args-BC by (simp add: join-sym l-BC-C)
        then have (map2 (⊔) As BCs)!i ≠ None
          unfolding nth-map[OF i] using i by auto
       }
       then show ?thesis by (simp add: list-all-length those-not-none-xs)
      qed
      then show ?thesis
       unfolding joinBC BC RuleA αβ unfolding join-opt.simps join.simps(7)
by force
    next
      case (RuleC γ Cs)
      from joinBC have βγ:β = γ and l-B-C:length Bs = length Cs
        using RuleB RuleC join-rule-rule by blast+
        obtain BCs where BC:BC = Prule β BCs and l-BC-B:length BCs =
length Bs
        and args-BC:(∀ i<length BCs. Bs!i ⊔  Cs!i = Some (BCs ! i))
```

```
          using join-rule-rule[OF joinBC[unfolded RuleB RuleC]] subterm(3,4)
RuleB(1) RuleC(1) by metis
        obtain ACs where AC:AC = Prule α ACs and l-AC-C:length ACs =
length Cs
        and args-AC:(∀ i<length ACs. As!i ⊔  Cs!i = Some (ACs ! i))
          using join-rule-rule[OF joinAC[unfolded RuleA RuleC]] subterm(2,4)
RuleA(1) RuleC(1) by metis
       have those (map2 (⊔) As BCs) ≠ None proof−
        {fix i assume i:i < length (zip As BCs)
         from RuleA(1,4) RuleB(1,4) RuleC(1,4) i have join-opt (As!i) ((Bs!i)
⊔ (Cs!i)) ≠ None
          using subterm(1) l-A-B l-B-C l-AC-C l-AB-B args-AB args-AC args-BC
           by (smt (verit) length-zip min-less-iff-conj nth-mem option.distinct(1)
supt.arg)
          then have (map2 (⊔) As BCs)!i ≠ None
           using i args-BC by simp
        }
        then show ?thesis
         by (simp add: list-all-length those-not-none-xs)
       qed
       then show ?thesis
        unfolding joinBC BC unfolding RuleA αβ join-opt.simps by (simp add:
option.case-eq-if)
     qed
   qed
  qed
qed

lemma join-list-defined:
  assumes ∀ a1 a2. a1 ∈ set As ∧ a2 ∈ set As ⟶ a1 ⊔ a2 ≠ None
    and ∀ a ∈ set As. a ∈ wf-pterm R and As ≠ []
  shows ∃ D. join-list As = Some D ∧ D ∈ wf-pterm R
using assms proof(induct length As arbitrary:As rule:full-nat-induct)
  case 1
  then show ?case proof(cases As rule:list.exhaust[case-names empty As])
    case (As A1 As')
    with 1 show ?thesis proof(cases As' rule:list.exhaust[case-names empty As'])
      case (As' A2 As'')
      have A1-wf:A1 ∈ wf-pterm R and A2-wf:A2 ∈ wf-pterm R
        using 1(3) unfolding As As' by auto
      from As' 1(2) obtain A12 where A12:A1 ⊔ A2 = Some A12
        unfolding As by fastforce
      with A1-wf A2-wf have A12-wf:A12 ∈ wf-pterm R
        by (simp add: join-wf-pterm)
      show ?thesis proof(cases As'' = [])
        case True
        show ?thesis
          unfolding As As' True join-list.simps using A12 A12-wf by simp
      next
```

```
      case False
      from 1 obtain D' where D':join-list As'' = Some D' D' ∈ wf-pterm R
          unfolding As As' by (metis False Suc-le-length-iff impossible-Cons
list.set-intros(2) nat-le-linear)
       from 1(2) have ∀ a1 a2. a1 ∈ set (A2 # As'') ∧ a2 ∈ set (A2 # As'')
⟶ a1 ⊔ a2 ≠ None
        unfolding As As' by force
      moreover have Suc (length (A2 # As'')) ≤ length As
        unfolding As As' by simp
      moreover have (∀ a∈set (A2 # As''). a ∈ wf-pterm R)
        using 1(3) unfolding As As' by simp
      moreover have A2 # As'' ≠ [] by simp
      ultimately obtain D where join-list (A2 # As'') = Some D and D-wf:D
∈ wf-pterm R
        using 1(1) by blast
      then have D:A2 ⊔ D' = Some D
        using D' False using join-list.elims by force
      moreover have A1 ⊔ D' ≠ None proof−
        from 1(2) have ∀ a1 a2. a1 ∈ set (A1 # As'') ∧ a2 ∈ set (A1 # As'')
⟶ a1 ⊔ a2 ≠ None
          unfolding As As' by force
        moreover have Suc (length (A1 # As'')) ≤ length As
          unfolding As As' by simp
        moreover have (∀ a∈set (A1 # As''). a ∈ wf-pterm R)
          using 1(3) unfolding As As' by simp
        moreover have A1 # As'' ≠ [] by simp
        ultimately have join-list (A1 # As'') ≠ None
          using 1(1) by (metis option.simps(3))
        with D' show ?thesis
          by (metis False join-list.simps(3) join-opt.simps(1) list.exhaust)
      qed
      moreover have A1 ⊔ A2 ≠ None
        using 1(2) unfolding As As' by simp
      ultimately have join-opt A1 (A2 ⊔ D') ≠ None
        using join-triple-defined D' A1-wf A2-wf unfolding join-list.simps by
blast
      moreover have join-list As = join-opt A1 (A2 ⊔ D')
          unfolding As As' using False by (metis D'(1) join-list.simps(3)
join-opt.simps(1) neq-Nil-conv)
      ultimately show ?thesis
       unfolding D join-opt.simps using D-wf A1-wf join-wf-pterm by fastforce
     qed
   qed simp
  qed simp
qed

lemma join-list-wf-pterm:
  assumes ∀ a ∈ set As. a ∈ wf-pterm R
    and join-list As = Some B
```

```
  shows B ∈ wf-pterm R
  using assms proof(induct As arbitrary:B)
  case (Cons A As)
  then show ?case proof(cases As = [])
    case True
    from Cons(2,3) show ?thesis unfolding join-list.simps True by simp
  next
    case False
    with Cons(3) obtain B' where B':join-list As = Some B'
      by (smt (verit, ccfv-threshold) join-list.elims join-opt.elims list.inject)
    with Cons have B' ∈ wf-pterm R
      by simp
    then show ?thesis using B' Cons
    by (metis False join-list.simps(3) join-opt.simps(1) join-wf-pterm list.set-intros(1)
neq-Nil-conv)
  qed
qed simp

lemma source-join-list:
  assumes join-list As = Some B and ∀ a ∈ set As. a ∈ wf-pterm R
  shows ⋀A. A ∈ set As ⟹ source A = source B
proof−
  fix Ai assume Ai ∈ set As
  then show co-initial Ai B using assms proof(induct As arbitrary: B)
    case Nil
    then show ?case by (simp add: source-join)
  next
    case (Cons A As)
    show ?case proof(cases As = [])
      case True
      from Cons show ?thesis unfolding True
        by (simp add: source-join)
    next
      case False
      have wf:A ∈ wf-pterm R ∀ a∈set As. a ∈ wf-pterm R
        using Cons(4) by simp-all
     from Cons(2,3)obtain B' where B':join-list As = Some B' join-list (A#As)
= join-opt A (Some B')
      by (metis False join-list.simps(3) join-opt.simps(2) list.exhaust option.exhaust)
      show ?thesis proof(cases Ai = A)
        case True
        show ?thesis unfolding True
             using B' Cons(3) False source-join wf by (metis join-list-wf-pterm
join-opt.simps(1))
      next
        case False
        then have Ai ∈ set As
          using Cons(2) by simp
        with Cons(1) B'(1) wf(2) have co-initial Ai B'
```

by *simp*
moreover from *B'(1) wf* **have** *B' ∈ wf-pterm R*
using *join-list-wf-pterm* **by** *blast*
ultimately show *?thesis*
by (*metis B'(2) Cons.prems(2) Residual-Join-Deletion.join-sym join-opt.simps(1) local.wf(1) source-join*)
qed
qed
qed
qed

end

3.3 Deletion

fun *deletion* :: ('*f*, '*v*) *pterm* ⇒ ('*f*,'*v*) *pterm* ⇒ ('*f*,'*v*) *pterm option* (**infixr** $-_p$ *70*)
where
Var x $-_p$ *Var y* =
(*if x = y then Some* (*Var x*) *else None*)
| *Pfun f As* $-_p$ *Pfun g Bs* =
(*if* (*f = g* ∧ *length As = length Bs*) *then*
(*case those* (*map2* ($-_p$) *As Bs*) *of*
Some xs ⇒ *Some* (*Pfun f xs*)
| *None* ⇒ *None*)
else None)
| *Prule* α *As* $-_p$ *Prule* β *Bs* =
(*if* α = β *then*
(*case those* (*map2* ($-_p$) *As Bs*) *of*
Some xs ⇒ *Some* ((*to-pterm* (*lhs* α)) · $\langle xs \rangle_\alpha$)
| *None* ⇒ *None*)
else None)
| *Prule* α *As* $-_p$ *B* =
(*case match B* (*to-pterm* (*lhs* α)) *of*
None ⇒ *None*
| *Some* σ ⇒
(*case those* (*map2* ($-_p$) *As* (*map* σ (*var-rule* α))) *of*
Some xs ⇒ *Some* (*Prule* α *xs*)
| *None* ⇒ *None*))
| *A* $-_p$ *B* = *None*

lemma *del-empty*:
assumes *A ∈ wf-pterm R*
shows *A* $-_p$ (*to-pterm* (*source A*)) = *Some A*
using *assms* **proof** (*induction A*)
case (*2 As f*)
then have *those* (*map2 deletion As* (*map* (*to-pterm* ∘ *source*) *As*)) = *Some As*
by (*simp add:those-some*)
then show *?case* **by** *simp*

next
case (*3* α *As*)
then have σ: *match* (*to-pterm* (*lhs* $\alpha \cdot \langle$*map source As*$\rangle_\alpha$)) (*to-pterm* (*lhs* α)) = *Some* ($\langle$*map* (*to-pterm* $\circ$ *source*) *As*$\rangle_\alpha$)
by (*metis* (*no-types*, *lifting*) *fun-mk-subst lhs-subst-trivial map-map to-pterm.simps*(*1*) *to-pterm-subst*)
from *3* **have** *those* (*map2 deletion As* (*map* (*to-pterm* $\circ$ *source*) *As*)) = *Some As*
by (*simp add*:*those-some*)
then have *args*:*those* (*map2 deletion As* (*map* ($\langle$*map* (*to-pterm* $\circ$ *source*) *As*$\rangle_\alpha$) (*var-rule* α))) = *Some As*
by (*metis 3.hyps*(*2*) *apply-lhs-subst-var-rule length-map*)
show *?case* **proof**(*cases source* (*Prule* α *As*))
case (*Var x*)
then show *?thesis*
using σ *residual.simps*(*4*)[*of* α *As x*] *args* **by** *auto*
next
case (*Fun f ts*)
then show *?thesis*
using σ *residual.simps*(*5*)[*of* α *As f*] *args* **by** *auto*
qed
qed *simp*

context *no-var-lhs*
begin
lemma *deletion-source*:
assumes *A* $\in$ *wf-pterm R B* $\in$ *wf-pterm R*
and *A* $-_p$ *B* = *Some C*
shows *source C* = *source A*
using *assms* **proof**(*induct A arbitrary*:*B C*)
case (*1 x*)
then show *?case* **proof** (*cases B*)
case (*1 y*)
then show *?thesis*
by (*metis 1.prems*(*2*) *deletion.simps*(*1*) *option.distinct*(*1*) *option.inject*)
next
case (*3* α *As*)
with *1 no-var-lhs* **show** *?thesis*
by *simp*
qed *simp*
next
case (*2 As f*)
then show *?case* **proof**(*cases B*)
case (*Pfun g Bs*)
from *2*(*3*) **have** *f*:*f* = *g*
unfolding *Pfun* **by** (*metis deletion.simps*(*2*) *not-None-eq*)
from *2*(*3*) **have** *l*:*length As* = *length Bs*
unfolding *Pfun* **by** (*metis deletion.simps*(*2*) *not-None-eq*)
from *2*(*3*) **obtain** *Cs* **where** *cs*:*those* (*map2* ($-_p$) *As Bs*) = *Some Cs*

```
    unfolding Pfun f using l by fastforce
  with 2(3) have c:C = Pfun g Cs
    unfolding Pfun by (simp add: f l)
  from cs l have l-cs:length Cs = length As
    using length-those by force
  {fix i assume i:i < length As
    with 2(2) have Bs!i ∈ wf-pterm R
      by (metis Pfun fun-well-arg l nth-mem)
    moreover from 2(3) i cs have As!i -p Bs!i = Some (Cs!i)
      using l those-some2 by fastforce
    ultimately have source (Cs!i) = source (As!i)
      using 2(1) using i nth-mem by blast
  }
  then show ?thesis unfolding c f
    using l-cs by (simp add: map-nth-eq-conv)
 qed simp-all
next
 case (3 α As)
 from 3(1) no-var-lhs obtain f ts where f:lhs α = Fun f ts
  by blast
 then show ?case proof(cases B)
  case (Var x)
  have match (Var x) (to-pterm (lhs α)) = None
   unfolding f by (smt (verit, ccfv-SIG) Term.term.simps(4) match-matches
not-Some-eq source.simps(1) source-to-pterm subst-apply-eq-Var)
  with 3(5) show ?thesis
    unfolding Var using f deletion.simps(4) by simp
 next
  case (Pfun g Bs)
  from 3(5) obtain σ where sigma:match B (to-pterm (lhs α)) = Some σ
    unfolding Pfun using deletion.simps(5) by fastforce
  with 3(5) obtain Cs where cs:those (map2 (-p) As (map σ (var-rule α))) =
Some Cs
    unfolding Pfun by fastforce
  with 3(5) have c:C = Prule α Cs
    using sigma unfolding Pfun by simp
  from cs 3(2) have l-cs:length Cs = length As
    using length-those by force
  {fix x assume x ∈ vars-term (lhs α)
    then obtain i where i:i < length (var-rule α) var-rule α !i = x
      by (metis in-set-conv-nth set-vars-term-list vars-term-list-vars-distinct)
    then have σ (var-rule α ! i) ∈ wf-pterm R
      using match-well-def[OF 3(4) sigma] by (metis vars-to-pterm)
    moreover from i cs have As!i -p σ (var-rule α ! i) = Some (Cs!i)
      using those-some2 3.hyps(2) by fastforce
    ultimately have source (Cs!i) = source (As!i)
      using 3(3) using i nth-mem 3.hyps(2) by force
    then have source ((⟨As⟩α) x) = source ((⟨Cs⟩α) x) using i
      by (metis 3.hyps(2) l-cs lhs-subst-var-i)
```

```
    }
    then show ?thesis unfolding c using l-cs 3(2) unfolding source.simps
    by (smt (verit, best) apply-lhs-subst-var-rule comp-def in-set-conv-nth length-map
nth-map set-remdups set-rev set-vars-term-list term-subst-eq-conv)
  next
    case (Prule β Bs)
    from 3(5) have alpha:α = β
      unfolding Prule by (metis deletion.simps(3) option.distinct(1))
    with 3 have l:length As = length Bs
      unfolding Prule using wf-pterm.cases by force
    from 3(5) obtain Cs where cs:those (map2 (−p) As Bs) = Some Cs
      unfolding Prule alpha by fastforce
    with 3(5) have c:C = to-pterm (lhs α) · ⟨Cs⟩α
      unfolding Prule alpha by simp
    from cs l have l-cs:length Cs = length As
      using length-those by force
    {fix i assume i:i < length As
      with 3(4) have Bs!i ∈ wf-pterm R
        unfolding Prule by (metis fun-well-arg l nth-mem)
      moreover from i cs have As!i −p Bs!i = Some (Cs!i)
        using l those-some2 by fastforce
      ultimately have source (Cs!i) = source (As!i)
        using 3(3) using i nth-mem by blast
    }
    then show ?thesis
      unfolding c using l-cs unfolding source.simps using source-apply-subst
    by (metis fun-mk-subst nth-map-conv source.simps(1) source-to-pterm to-pterm-wf-pterm)
  qed
qed
end
```

3.4 Computations With Single Redexes

When a proof term contains only a single rule symbol, we say it is a **single redex*.

```
definition ll-single-redex :: ('f, 'v) term ⇒ pos ⇒ ('f, 'v) prule ⇒ ('f, 'v) pterm
  where ll-single-redex s p α = (ctxt-of-pos-term p (to-pterm s))⟨Prule α (map
(to-pterm ∘ (λpi. s|-(p@pi))) (var-poss-list (lhs α)))⟩
```

The *ll* in *ll-single-redex* stands for **left−linear*, since this definition only makes sense for left-linear rules.

```
lemma source-single-redex:
  assumes p ∈ poss s
  shows source (ll-single-redex s p α) = (ctxt-of-pos-term p s)⟨(lhs α) · ⟨map (λpi.
s|-(p@pi)) (var-poss-list (lhs α))⟩α⟩
proof−
  have source (Prule α (map (to-pterm ∘ (λpi. s|-(p@pi))) (var-poss-list (lhs α))))
= (lhs α) · ⟨map (λpi. s|-(p@pi)) (var-poss-list (lhs α))⟩α
```

unfolding *source.simps* **using** *map-nth-eq-conv* **by** *fastforce*
with *assms* **show** *?thesis*
unfolding *ll-single-redex-def* **by** (*metis context-source source-to-pterm to-pterm-ctxt-of-pos-apply-term*)

qed

lemma *target-single-redex*:
assumes $p \in$ *poss s*
shows *target* (*ll-single-redex s p* α) = (*ctxt-of-pos-term p s*)$\langle$(*rhs* α) $\cdot$ $\langle$*map* (λ*pi*. *s*|-(*p*@*pi*)) (*var-poss-list* (*lhs* α))$\rangle_\alpha\rangle$
proof−
have *target* (*Prule* α (*map* (*to-pterm* $\circ$ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α)))) = (*rhs* α) $\cdot$ $\langle$*map* (λ*pi*. *s*|-(*p*@*pi*)) (*var-poss-list* (*lhs* α))$\rangle_\alpha$
unfolding *target.simps* **by** (*metis* (*no-types*, *lifting*) *fun-mk-subst map-map target-empty-apply-subst target-to-pterm to-pterm.simps*(*1*) *to-pterm-empty to-pterm-subst*)
with *assms* **show** *?thesis*
unfolding *ll-single-redex-def* **using** *target-to-pterm-ctxt to-pterm-ctxt-at-pos*
by *metis*
qed

lemma *single-redex-rstep*:
assumes *to-rule* $\alpha \in R$ $p \in$ *poss s*
shows (*source* (*ll-single-redex s p* α), *target* (*ll-single-redex s p* α)) $\in$ *rstep R*
using *source-single-redex target-single-redex assms* **by** *blast*

lemma *single-redex-neq*:
assumes (α, *p*) $\neq$ (β, *q*) $p \in$ *poss s* $q \in$ *poss s*
shows *ll-single-redex s p* $\alpha \neq$ *ll-single-redex s q* β
proof−
from *assms*(*1*) **consider** $\alpha \neq \beta \wedge p = q \mid p \neq q$
by *fastforce*
then show *?thesis* **proof**(*cases*)
case *1*
then have *Prule* α (*map* (*to-pterm* $\circ$ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α))) $\neq$ *Prule* β (*map* (*to-pterm* $\circ$ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α)))
by *simp*
with *1* **show** *?thesis*
using *assms*(*2*,*3*) **unfolding** *ll-single-redex-def* **by** *simp*
next
case *2*
show *?thesis* **proof**(*cases p* $\in$ *poss* (*ll-single-redex s q* β))
case *True*
from *2* **consider** (*qp*) $q <_p p \mid$ (*par*) $q \perp p \mid$ (*pq*) $p <_p q$
using *pos-cases* **by** *force*
then show *?thesis* **proof**(*cases*)
case *qp*
then obtain *i r* **where** *r*:*q*@(*i*#*r*) = *p*
using *less-pos-def'* **by** (*metis neq-Nil-conv*)
with ‹*p* $\in$ *poss* (*ll-single-redex s q* β)› **have** *i*#*r* $\in$ *poss* (*Prule* β (*map*

(*to-pterm* ∘ (λ*pi*. *s* |- (*q* @ *pi*))) (*var-poss-list* (*lhs* β))))
unfolding *ll-single-redex-def* **using** *assms*(*3*) **by** (*metis hole-pos-ctxt-of-pos-term hole-pos-poss-conv poss-list-sound poss-list-to-pterm*)
then have *i*:*i* < *length* (*var-poss-list* (*lhs* β)) **and** *r* ∈ *poss* (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*q* @ *pi*))) (*var-poss-list* (*lhs* β))!*i*)
by *auto*
then have *r* ∈ *poss* (*to-pterm* (*s* |- (*q* @ (*var-poss-list* (*lhs* β)!*i*))))
by *simp*
then obtain *s*′ **where** (*Prule* β (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*q* @ *pi*))) (*var-poss-list* (*lhs* β))))|-(*i*#*r*) = *to-pterm s*′
by (*metis* (*no-types*, *lifting*) *comp-apply ctxt-supt-id i nth-map poss-list-sound poss-list-to-pterm subt-at.simps*(*2*) *subt-at-hole-pos to-pterm-ctxt-of-pos-apply-term*)

then have (*Prule* β (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*q* @ *pi*))) (*var-poss-list* (*lhs* β))))|-(*i*#*r*) ≠ *Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α)))
using *to-pterm.elims* **by** *auto*
then show *?thesis* **using** *r assms*(*2*,*3*) **unfolding** *ll-single-redex-def*
by (*smt* (*verit*, *del-insts*) *hole-pos-ctxt-of-pos-term hole-pos-poss p-in-poss-to-pterm replace-at-subt-at subt-at-append*)
next
case *par*
then have *ll-single-redex s q* β |- *p* = *to-pterm s* |-*p*
using *True* **unfolding** *ll-single-redex-def*
by (*simp add*: *assms*(*2*,*3*) *p-in-poss-to-pterm parallel-replace-at-subt-at*)
then show *?thesis*
using *assms*(*2*,*3*) **unfolding** *ll-single-redex-def*
by (*metis ctxt-supt-id hole-pos-ctxt-of-pos-term is-empty-step.simps*(*3*) *p-in-poss-to-pterm subt-at-hole-pos to-pterm-ctxt-of-pos-apply-term to-pterm-empty*)
next
case *pq*
then obtain *r* **where** *r*:*q* = *p*@*r r* ≠ []
using *less-pos-def*′ **by** *blast*
then have ∗:*ll-single-redex s q* β |- *p* = (*ctxt-of-pos-term r* (*to-pterm* (*s*|-*p*)))⟨*Prule* β (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*q* @ *pi*))) (*var-poss-list* (*lhs* β)))⟩
using *True* **unfolding** *ll-single-redex-def r* **by** (*metis* (*no-types*, *lifting*) *assms*(*2*) *ctxt-apply-subt-at ctxt-supt-id p-in-poss-to-pterm replace-at-subt-at to-pterm-ctxt-of-pos-apply-term*)
from *r*(*2*) *assms*(*3*) **obtain** *f ts i r*′ **where** *f*:*s*|-*p* = *Fun f ts* **and** *r*′:*r* = *i*#*r*′
unfolding *r* **by** (*metis args-poss neq-Nil-conv poss-append-poss*)
have *ll-single-redex s q* β |- *p* ≠ *Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α)))
unfolding ∗ **unfolding** *ll-single-redex-def f to-pterm.simps r*′ *ctxt-of-pos-term.simps intp-actxt.simps* **by** *simp*
then show *?thesis*
using *assms*(*2*) **unfolding** *ll-single-redex-def*
by (*metis p-in-poss-to-pterm replace-at-subt-at*)
qed

```
    next
      case False
      then show ?thesis unfolding ll-single-redex-def using assms(2)
        by (metis hole-pos-ctxt-of-pos-term hole-pos-poss p-in-poss-to-pterm)
    qed
  qed
qed

context left-lin-wf-trs
begin
lemma rstep-exists-single-redex:
  assumes (s, t) ∈ rstep R
  shows ∃ A p α. A = (ll-single-redex s p α) ∧ source A = s ∧ target A = t  ∧
to-rule α ∈ R ∧ p ∈ poss s
proof−
  from assms obtain C σ l r where lr:(l, r) ∈ R and s:s = C⟨l · σ⟩ and t: t =
C⟨r · σ⟩
    by blast
  from s obtain p where p:p ∈ poss s and C:C = ctxt-of-pos-term p s
    using hole-pos-poss by fastforce
  let ?subst=⟨map (λpi. s |- (p @ pi)) (var-poss-list l)⟩(l →r)
  {fix x assume x ∈ vars-term l
    then obtain i where i:i < length (vars-term-list l) vars-term-list l ! i = x
      by (metis in-set-idx set-vars-term-list)
    with left-lin lr have var-l:vars-distinct l ! i = x
      using linear-term-var-vars-term-list left-linear-trs-def by fastforce
    let ?p=var-poss-list l ! i
    from i have l |- ?p = Var x using vars-term-list-var-poss-list by auto
    moreover have l · σ = s|-p using s C p replace-at-subt-at by fastforce
    ultimately have left:σ x = (s |-p) |- ?p
      by (metis eval-term.simps(1) i(1) length-var-poss-list nth-mem subt-at-subst
var-poss-imp-poss var-poss-list-sound)
    from i have map (λpi. s |- (p @ pi)) (var-poss-list l) !i = (s |-p) |- ?p
      by (simp add: length-var-poss-list p)
    with left var-l have σ x = ?subst x unfolding mk-subst-def prule.sel
    by (smt (verit, best) case-prodE comp-apply distinct-rev i(1) left-lin left-linear-trs-def
length-map length-var-poss-list linear-term-var-vars-term-list lr mk-subst-def mk-subst-distinct
prod.sel(1) remdups-id-iff-distinct rev-rev-ident)
  }note subst=this
   then have (ctxt-of-pos-term p s)⟨l · ⟨map (λpi. s |- (p @ pi)) (var-poss-list
l)⟩(l→r)⟩ = C⟨l · σ⟩
    using C by (simp add: eval-same-vars)
  then have source (ll-single-redex s p (Rule l r)) = s
    using source-single-redex[OF p] s by auto
  moreover have target (ll-single-redex s p (l → r)) = t
   using subst varcond lr target-single-redex[OF p] eval-same-vars-cong unfolding
t C
    by (smt (verit) case-prodD prule.sel(1) prule.sel(2) vars-term-subset-subst-eq)
  ultimately show ?thesis using lr p by fastforce
```

qed
end

lemma *single-redex-wf-pterm*:
  assumes *to-rule* $\alpha \in R$ **and** *lin*:*linear-term* (*lhs* α)
    and *p*:$p \in$ *poss s*
  shows *ll-single-redex s p* $\alpha \in$ *wf-pterm R*
proof−
  from *lin* **have** *l*:*length* (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α))) = *length* (*var-rule* α)
    using *length-var-poss-list linear-term-var-vars-term-list* **by** *fastforce*
  have *Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α))) ∈ *wf-pterm R*
    using *wf-pterm.intros*(*3*)[*OF assms*(*1*) *l*] *to-pterm-wf-pterm* **by** *force*
  then show *?thesis* **unfolding** *ll-single-redex-def*
    using *ctxt-wf-pterm p to-pterm-wf-pterm* **by** (*metis p-in-poss-to-pterm*)
qed

Interaction of a single redex Δ, contained in A with the proof term A.

locale *single-redex* = *left-lin-no-var-lhs* +
  fixes *A* Δ *p q* α
  assumes *a-well*:*A* ∈ *wf-pterm R*
    and *p*:$p \in$ *poss* (*source A*) **and** *q*:$q \in$ *poss A*
    and *pq*:*ctxt-of-pos-term p* (*source A*) = *source-ctxt* (*ctxt-of-pos-term q A*)
    and *delta*:Δ = *ll-single-redex* (*source A*) *p* α
    and *aq*:*A*|-*q* = *Prule* α (*map* (λ*i*. *A*|-(*q*@[*i*])) [*0*..<*length* (*var-rule* α)])
begin

interpretation *residual-op*:*op-proof-term R residual*
  using *op-proof-term.intro*[*OF left-lin-no-var-lhs-axioms*] *op-proof-term-axioms.intro*[*of R residual*] *res-empty2* **by** *force*

interpretation *deletion-op*:*op-proof-term R deletion*
  using *op-proof-term.intro*[*OF left-lin-no-var-lhs-axioms*] *op-proof-term-axioms.intro*[*of R deletion*] *del-empty* **by** *force*

abbreviation *As* ≡ (*map* (λ*i*. *A*|-(*q*@[*i*])) [*0*..<*length* (*var-rule* α)])

lemma *length-as*:*length As* = *length* (*var-rule* α)
  by *simp*

lemma *as-well*:∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
  using *subt-at-is-wf-pterm a-well aq q*
  by (*metis fun-well-arg nth-mem*)

lemma *a*:*A* = (*ctxt-of-pos-term q A*)⟨*Prule* α *As*⟩
  using *aq* **by** (*simp add*: *q replace-at-ident*)

lemma *rule-in-TRS*: *to-rule* $\alpha \in R$

proof−
from *a-well a q* **have** *Prule α As* ∈ *wf-pterm R*
by (*metis subt-at-ctxt-of-pos-term subt-at-is-wf-pterm*)
then show *?thesis*
using *wf-pterm.cases* **by** *force*
qed

lemma *lin-lhs*:*linear-term* (*lhs α*)
using *rule-in-TRS left-lin left-linear-trs-def* **by** *fastforce*

lemma *source-at-pq*:*source* (*A*|-*q*) = (*source A*)|-*p*
proof−
from *a-well q* **have** (*ctxt-of-pos-term q A*) ∈ *wf-pterm-ctxt R*
by (*simp add*: *ctxt-of-pos-term-well*)
then have *source A* = (*source-ctxt* (*ctxt-of-pos-term q A*)) ⟨*source* (*A*|-*q*)⟩
using *source-ctxt-apply-term q* **by** (*metis ctxt-supt-id*)
moreover from *p* **have** *source A* = (*ctxt-of-pos-term p* (*source A*)) ⟨(*source A*)|-*p*⟩
by (*simp add*: *replace-at-ident*)
ultimately show *?thesis*
using *pq p q* **by** *simp*
qed

lemma *single-redex-pterm*:
shows Δ = (*ctxt-of-pos-term p* (*to-pterm* (*source A*)))⟨*Prule α* (*map* (*to-pterm* ∘ *source*) *As*)⟩
proof−
from *lin-lhs* **have** *l2*:*length* (*var-poss-list* (*lhs α*)) = *length* (*var-rule α*)
by (*metis length-var-poss-list linear-term-var-vars-term-list*)
{fix *i* **assume** *i*:*i* < *length* (*var-poss-list* (*lhs α*))
let *?pi*=*var-poss-list* (*lhs α*)!*i*
from *i* **have** ∗:(*lhs α*)|-*?pi* = *Var* ((*var-rule α*)!*i*)
using *lin-lhs* **by** (*metis linear-term-var-vars-term-list length-var-poss-list vars-term-list-var-poss-list*)
from *source-at-pq* **have** *source A* |- (*p* @ *?pi*) = *source* (*Prule α As*)|-*?pi*
by (*metis a p q subt-at-append subt-at-ctxt-of-pos-term*)
also have ... = *Var* ((*var-rule α*)!*i*) · ⟨*map source As*⟩$_α$
unfolding *source.simps* **using** *subt-at-subst* ∗ *i nth-mem var-poss-imp-poss* **by** *fastforce*
also have ... = *source* (*As*!*i*)
unfolding *eval-term.simps* **using** *i lhs-subst-var-i length-as l2* **by** (*metis* (*no-types*, *lifting*) *length-map nth-map*)
finally have *source A* |- (*p* @ *?pi*) = *source* (*As*!*i*) **.**
}
with *l2* **show** *?thesis*
unfolding *delta ll-single-redex-def* **by** (*simp add*: *nth-map-conv*)
qed

lemma *delta-trs-wf-pterm*:

shows $\Delta \in$ *wf-pterm R*
proof−
have *well2*:*Prule* α (*map* (*to-pterm* ∘ *source*) *As*) ∈ *wf-pterm R* **proof**−
from *a-well a q* **have** *Prule* α *As* ∈ *wf-pterm R*
by (*metis subt-at-ctxt-of-pos-term subt-at-is-wf-pterm*)
then have *to-rule* $\alpha \in R$
using *wf-pterm.cases* **by** *auto*
then show *?thesis*
by (*simp add*: *wf-pterm.intros*(*3*))
qed
show *?thesis* **unfolding** *single-redex-pterm*
using *p well2* **by** (*simp add*: *p-in-poss-to-pterm ctxt-wf-pterm*)
qed

lemma *source-delta*: *source* Δ = *source A*
proof−
have *src*:*source* (*Prule* α (*map* (*to-pterm* ∘ *source*) *As*)) = *source* (*Prule* α *As*)
unfolding *source.simps* **by** (*metis* (*no-types*, *lifting*) *comp-eq-dest-lhs list.map-comp list.map-cong0 source-to-pterm*)
moreover have *source-ctxt* (*ctxt-of-pos-term p* (*to-pterm* (*source A*))) = *source-ctxt* (*ctxt-of-pos-term q A*)
using *pq* **by** (*metis p source-to-pterm-ctxt′ to-pterm-ctxt-at-pos*)
ultimately show *?thesis* **unfolding** *single-redex-pterm*
using *p p q* **by** (*metis aq p-in-poss-to-pterm pq replace-at-ident source-at-pq source-ctxt-apply-term to-pterm-trs-ctxt*)
qed

lemma *residual*:
shows *A re* Δ = *Some* ((*ctxt-of-pos-term q A*)⟨(*to-pterm* (*rhs* α)) · ⟨*As*⟩$_\alpha$⟩)
proof−
have *l*:*length* (*map2* (*re*) *As* (*map* (*to-pterm* ∘ *source*) *As*)) = *length As*
by *simp*
{fix *i* **assume** *i*:*i* < *length As*
with *as-well* **have** *As*!*i re* (*to-pterm* ∘ *source*) (*As*!*i*) = *Some* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *o-apply res-empty2*)
then have *map2* (*re*) *As* (*map* (*to-pterm* ∘ *source*) *As*) ! *i* = *Some* (*As* ! *i*)
using *i* **by** *force*
}
then have ∗:*those* (*map2* (*re*) *As* (*map* (*to-pterm* ∘ *source*) *As*)) = *Some As*
using *those-some*[*OF l*] **using** *l* **by** *presburger*
then have (*Prule* α *As*) *re* (*Prule* α (*map* (*to-pterm* ∘ *source*) *As*)) = *Some* ((*to-pterm* (*rhs* α)) · ⟨*As*⟩$_\alpha$)
using *residual.simps*(*3*)[*of* α *As* α (*map* (*to-pterm* ∘ *source*) *As*)] **by** *simp*
moreover from *single-redex-pterm* **have** Δ = (*to-pterm-ctxt* (*source-ctxt* (*ctxt-of-pos-term q A*)))⟨(*Prule* α (*map* (*to-pterm* ∘ *source*) *As*))⟩
unfolding *delta ll-single-redex-def pq*[*symmetric*] **by** (*simp add*: *p to-pterm-ctxt-at-pos*)
ultimately show *?thesis*
using *a residual-op.apply-f-ctxt* **by** (*metis a-well ctxt-of-pos-term-well q*)
qed

lemma *residual-well*:
the (*A re* Δ) ∈ *wf-pterm R*
using *a-well* **by** (*metis delta-trs-wf-pterm option.sel residual residual-well-defined*)

lemma *target-residual*:*target* (*the* (*A re* Δ)) = *target A*
apply(*subst* (*2*) *a*)
unfolding *residual option.sel*
apply(*subst* (*1 2*) *context-target*)
by (*metis fun-mk-subst target.simps*(*1*) *target.simps*(*3*) *target-empty-apply-subst target-to-pterm to-pterm-empty*)

lemma *deletion*:
shows *A* −$_p$ Δ = *Some* ((*ctxt-of-pos-term q A*)⟨(*to-pterm* (*lhs* α)) · ⟨*As*⟩$_\alpha$⟩)
proof−
have *l*:*length* (*map2* (−$_p$) *As* (*map* (*to-pterm* ∘ *source*) *As*)) = *length As*
by *simp*
{fix *i* **assume** *i*:*i* < *length As*
with *as-well* **have** *As*!*i* −$_p$ (*to-pterm* ∘ *source*) (*As*!*i*) = *Some* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *o-apply del-empty*)
then have *map2* (−$_p$) *As* (*map* (*to-pterm* ∘ *source*) *As*) ! *i* = *Some* (*As* ! *i*)
using *i* **by** *force*
}
then have ∗:*those* (*map2* (−$_p$) *As* (*map* (*to-pterm* ∘ *source*) *As*)) = *Some As*
using *those-some*[*OF l*] **using** *l* **by** *presburger*
then have (*Prule* α *As*) −$_p$ (*Prule* α (*map* (*to-pterm* ∘ *source*) *As*)) = *Some* ((*to-pterm* (*lhs* α)) · ⟨*As*⟩$_\alpha$)
using *deletion.simps*(*3*)[*of* α *As* α (*map* (*to-pterm* ∘ *source*) *As*)] **by** *simp*
moreover from *single-redex-pterm* **have** Δ = (*to-pterm-ctxt* (*source-ctxt* (*ctxt-of-pos-term q A*)))⟨(*Prule* α (*map* (*to-pterm* ∘ *source*) *As*))⟩
unfolding *delta ll-single-redex-def pq*[*symmetric*] **by** (*simp add*: *p to-pterm-ctxt-at-pos*)
ultimately show *?thesis*
using *a deletion-op.apply-f-ctxt* **by** (*metis a-well ctxt-of-pos-term-well q*)
qed

lemma *deletion-well*:
shows *the* (*A* −$_p$ Δ) ∈ *wf-pterm R*
proof−
have ∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R*
by (*metis a a-well fun-well-arg q subt-at-ctxt-of-pos-term subt-at-is-wf-pterm*)
then have *to-pterm* (*lhs* α) · ⟨*As*⟩$_\alpha$ ∈ *wf-pterm R*
by (*meson lhs-subst-well-def nth-mem to-pterm-wf-pterm*)
then show *?thesis* **unfolding** *deletion option.sel*
by (*simp add*: *a-well ctxt-wf-pterm q*)
qed

end

locale *single-redex′* = *left-lin-wf-trs* +
  fixes A Δ p q α σ
  assumes *a-well*:$A \in$ *wf-pterm* R **and** *rule-in-TRS*:*to-rule* $\alpha \in R$
    and *p*:$p \in$ *poss* (*source* A) **and** *q*:$q \in$ *poss* A
    and *pq*:*ctxt-of-pos-term* p (*source* A) = *source-ctxt* (*ctxt-of-pos-term* q A)
    and *delta*:Δ = *ll-single-redex* (*source* A) p α
    and *aq*:$A|$-q = (*to-pterm* (*lhs* α)) $\cdot$ σ
begin

interpretation *residual-op*:*op-proof-term* R *residual* **proof**−
  have ∗:*left-lin-no-var-lhs* R
    by (*simp add*: *left-lin-axioms left-lin-no-var-lhs.intro no-var-lhs-axioms*)
  then show *op-proof-term* R (*re*)
   using *op-proof-term.intro*[*OF* ∗] *op-proof-term-axioms.intro*[*of R residual*] *res-empty2*
by *force*
qed

lemma *a*:A = (*ctxt-of-pos-term* q A)⟨(*to-pterm* (*lhs* α)) $\cdot$ σ⟩
  using *aq* **by** (*simp add*: *q replace-at-ident*)

lemma *lin-lhs*:*linear-term* (*lhs* α)
  using *rule-in-TRS left-lin left-linear-trs-def* **by** *fastforce*

lemma *is-fun-lhs*:*is-Fun* (*lhs* α)
  using *rule-in-TRS* **using** *no-var-lhs* **by** *blast*

abbreviation $As \equiv$ *map* σ (*var-rule* α)

lemma *lhs-subst*: (*to-pterm* (*lhs* α)) $\cdot$ σ = (*to-pterm* (*lhs* α)) $\cdot$ $\langle As \rangle_\alpha$
proof−
  {fix x **assume** $x \in$ *vars-term* (*to-pterm* (*lhs* α))
    then obtain i **where** x = *var-rule* α!i **and** $i <$ *length* (*var-rule* α)
    by (*metis in-set-idx set-vars-term-list vars-term-list-vars-distinct vars-to-pterm*)
    then have σ x = ($\langle As \rangle_\alpha$) x
      by (*metis* (*mono-tags*, *lifting*) *apply-lhs-subst-var-rule length-map nth-map*)
  }
  then show *?thesis*
    using *term-subst-eq-conv* **by** *blast*
qed

lemma *rhs-subst*: (*to-pterm* (*rhs* α)) $\cdot$ σ = (*to-pterm* (*rhs* α)) $\cdot$ $\langle As \rangle_\alpha$
proof−
  {fix x **assume** $x \in$ *vars-term* (*to-pterm* (*rhs* α))
    then have $x \in$ *vars-term* (*to-pterm* (*lhs* α))
     using *no-var-lhs varcond rule-in-TRS set-vars-term-list subsetD vars-to-pterm*
by (*metis case-prodD*)
    then obtain i **where** x = *var-rule* α!i **and** $i <$ *length* (*var-rule* α)
    by (*metis in-set-idx set-vars-term-list vars-term-list-vars-distinct vars-to-pterm*)
    then have σ x = ($\langle As \rangle_\alpha$) x

by (*metis* (*mono-tags*, *lifting*) *apply-lhs-subst-var-rule length-map nth-map*)
}
then show *?thesis*
using *term-subst-eq-conv* **by** *blast*
qed

lemma *as-well*:$\forall\, i < \mathit{length}\ \mathit{As}.\ \mathit{As}!i \in \mathit{wf\text{-}pterm}\ R$
using *a-well aq* **by** (*metis length-map lhs-subst lhs-subst-args-wf-pterm nth-mem q subt-at-is-wf-pterm*)

lemma *source-at-pq*:$\mathit{source}\ (A|\text{-}q) = (\mathit{source}\ A)|\text{-}p$
proof−
from *a-well q* **have** $(\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ q\ A) \in \mathit{wf\text{-}pterm\text{-}ctxt}\ R$
by (*simp add*: *ctxt-of-pos-term-well*)
then have $\mathit{source}\ A = (\mathit{source\text{-}ctxt}\ (\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ q\ A))\ \langle \mathit{source}\ (A|\text{-}q)\rangle$
using *source-ctxt-apply-term q* **by** (*metis ctxt-supt-id*)
moreover from *p* **have** $\mathit{source}\ A = (\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ p\ (\mathit{source}\ A))\ \langle(\mathit{source}\ A)|\text{-}p\rangle$
by (*simp add*: *replace-at-ident*)
ultimately show *?thesis*
using *pq p q* **by** *simp*
qed

lemma *single-redex-pterm*:
shows $\Delta = (\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ p\ (\mathit{to\text{-}pterm}\ (\mathit{source}\ A)))\langle \mathit{Prule}\ \alpha\ (\mathit{map}\ (\mathit{to\text{-}pterm} \circ \mathit{source})\ \mathit{As})\rangle$
proof−
from *lin-lhs* **have** *l2*:$\mathit{length}\ (\mathit{var\text{-}poss\text{-}list}\ (\mathit{lhs}\ \alpha)) = \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$
by (*metis length-var-poss-list linear-term-var-vars-term-list*)
{fix *i* **assume** *i*:$i < \mathit{length}\ (\mathit{var\text{-}poss\text{-}list}\ (\mathit{lhs}\ \alpha))$
let *?pi*=$\mathit{var\text{-}poss\text{-}list}\ (\mathit{lhs}\ \alpha)!i$
from *i* **have** $*$:$(\mathit{lhs}\ \alpha)|\text{-}?pi = \mathit{Var}\ ((\mathit{var\text{-}rule}\ \alpha)!i)$
using *lin-lhs* **by** (*metis linear-term-var-vars-term-list length-var-poss-list vars-term-list-var-poss-list*)
from *source-at-pq* **have** $\mathit{source}\ A\ |\text{-}\ (p\ @\ ?pi) = \mathit{source}\ ((\mathit{to\text{-}pterm}\ (\mathit{lhs}\ \alpha)) \cdot \langle As\rangle_\alpha)|\text{-}?pi$
using *lhs-subst* **by** (*metis a p q subt-at-append subt-at-ctxt-of-pos-term*)
also have $\ldots = \mathit{Var}\ ((\mathit{var\text{-}rule}\ \alpha)!i) \cdot \langle \mathit{map}\ \mathit{source}\ \mathit{As}\rangle_\alpha$
using *subt-at-subst* $*$
by (*metis* (*no-types*, *lifting*) *fun-mk-subst i nth-mem source.simps*(*1*) *source-apply-subst source-to-pterm to-pterm-wf-pterm var-poss-imp-poss var-poss-list-sound*)
also have $\ldots = \mathit{source}\ (\mathit{As}!i)$
unfolding *eval-term.simps* **using** *i lhs-subst-var-i l2* **by** (*metis* (*no-types*, *lifting*) *length-map nth-map*)
finally have $\mathit{source}\ A\ |\text{-}\ (p\ @\ ?pi) = \mathit{source}\ (\mathit{As}!i)$ **.**
}
with *l2* **show** *?thesis*
unfolding *delta ll-single-redex-def* **by** (*simp add*: *map-eq-conv*′)
qed

lemma *residual*:
shows $A\ re\ \Delta = Some\ ((ctxt\text{-}of\text{-}pos\text{-}term\ q\ A)\langle(to\text{-}pterm\ (rhs\ \alpha))\cdot\sigma\rangle)$
proof−
have *l*:$length\ (map2\ (re)\ As\ (map\ (to\text{-}pterm \circ source)\ As)) = length\ As$
by *simp*
{fix i **assume** i:$i < length\ As$
with *as-well* **have** $As!i\ re\ (to\text{-}pterm \circ source)\ (As!i) = Some\ (As!i)$
by (*metis comp-apply res-empty2*)
then have $map2\ (re)\ As\ (map\ (to\text{-}pterm \circ source)\ As)\ !\ i = Some\ (As\ !\ i)$
using i **by** *force*
}
then have $*$:$those\ (map2\ (re)\ As\ (map\ (to\text{-}pterm \circ source)\ As)) = Some\ As$
using *those-some*[*OF l*] **using** *l* **by** *presburger*
from *is-fun-lhs* **obtain** $f\ As'$ **where** f:$(to\text{-}pterm\ (lhs\ \alpha)\cdot\langle As\rangle_\alpha) = Pfun\ f\ As'$
by *fastforce*
then have *match*:$match\ (Pfun\ f\ As')\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ (\langle As\rangle_\alpha)$
by (*metis lhs-subst-trivial*)
have *map*:$map\ (\langle As\rangle_\alpha)\ (var\text{-}rule\ \alpha) = As$
using *apply-lhs-subst-var-rule length-map* **by** *blast*
have $((to\text{-}pterm\ (lhs\ \alpha))\cdot\sigma)\ re\ (Prule\ \alpha\ (map\ (to\text{-}pterm \circ source)\ As)) = Some$
$((to\text{-}pterm\ (rhs\ \alpha))\cdot\sigma)$
unfolding *rhs-subst lhs-subst* **using** *residual.simps*(7)[*of f As'* α (*map* (*to-pterm*
$\circ$ *source*) *As*)]
unfolding *match f* **using** $*$ *map* **by** (*metis option.simps*(5))
moreover from *single-redex-pterm* **have** $\Delta = (to\text{-}pterm\text{-}ctxt\ (source\text{-}ctxt\ (ctxt\text{-}of\text{-}pos\text{-}term$
$q\ A)))\langle(Prule\ \alpha\ (map\ (to\text{-}pterm \circ source)\ As))\rangle$
unfolding *delta ll-single-redex-def pq*[*symmetric*] **by** (*simp add*: *p to-pterm-ctxt-at-pos*)
ultimately show *?thesis*
using *a residual-op.apply-f-ctxt* **by** (*metis a-well ctxt-of-pos-term-well q*)
qed

end

end

4 Orthogonal Proof Terms

theory *Orthogonal-PT*
imports
Residual-Join-Deletion
begin

inductive *orthogonal*::('*f*, '*v*) *pterm* ⇒ ('*f*, '*v*) *pterm* ⇒ *bool* (**infixl** $\perp_p$ *50*)
where
$Var\ x \perp_p Var\ x$
| $length\ As = length\ Bs \Longrightarrow \forall\ i < length\ As.\ As!i \perp_p Bs!i \Longrightarrow Fun\ f\ As \perp_p Fun\ f$
Bs
| $length\ As = length\ Bs \Longrightarrow \forall (a,b) \in set(zip\ As\ Bs).\ a \perp_p b \Longrightarrow (Prule\ \alpha\ As) \perp_p$

(*to-pterm* (*lhs* α)) $\cdot$ $\langle Bs \rangle_\alpha$
| *length As* = *length Bs* $\Longrightarrow$ $\forall$ (*a*,*b*) $\in$ *set*(*zip As Bs*). *a* $\perp_p$ *b* $\Longrightarrow$ (*to-pterm* (*lhs* α)) $\cdot$ $\langle As \rangle_\alpha$ $\perp_p$ (*Prule* α *Bs*)
lemmas *orthogonal.intros*[*intro*]

lemma *orth-symp*: *symp* ($\perp_p$)
proof
  {fix *A B*::('*f*, '*v*) *pterm* **assume** *A* $\perp_p$ *B*
    then show *B* $\perp_p$ *A* **proof**(*induct*)
      case (*3 As Bs* α)
       then show *?case* **using** *orthogonal.intros*(*4*)[**where** α=α **and** *Bs*=*As* **and** *As*=*Bs*]
        using *zip-symm* **by** *fastforce*
    next
      case (*4 As Bs* α)
       then show *?case* **using** *orthogonal.intros*(*3*)[**where** α=α **and** *As*=*Bs* **and** *Bs*=*As*]
        using *zip-symm* **by** *fastforce*
    qed (*simp-all add*:*orthogonal.intros*)
  }
qed

lemma *equal-imp-orthogonal*:
  shows *A* $\perp_p$ *A*
  by(*induct A*) (*simp-all add*: *orthogonal.intros*)

lemma *source-orthogonal*:
  assumes *source A* = *t*
  shows *A* $\perp_p$ *to-pterm t*
  using *assms* **proof**(*induct A arbitrary*:*t*)
  case (*Prule* α *As*)
  then have *t*:*to-pterm t* = (*to-pterm* (*lhs* α)) $\cdot$ $\langle$*map* (*to-pterm* $\circ$ *source*) *As*$\rangle_\alpha$
   by (*metis fun-mk-subst list.map-comp source.simps*(*3*) *to-pterm.simps*(*1*) *to-pterm-subst*)
  from *Prule*(*1*) **have** $\forall$ (*a*,*b*) $\in$ *set* (*zip As* (*map* (*to-pterm* $\circ$ *source*) *As*)). *a* $\perp_p$ *b*
      by (*metis* (*mono-tags*, *lifting*) *case-prod-beta*′ *comp-def in-set-zip nth-map zip-fst*)
  with *t* **show** *?case*
    using *orthogonal.intros*(*3*) **by** (*metis length-map*)
qed (*simp-all add*:*orthogonal.intros*)

lemma *orth-imp-co-initial*:
  assumes *A* $\perp_p$ *B*
  shows *co-initial A B*
  using *assms* **proof**(*induct rule*: *orthogonal.induct*)
  case (*2 As Bs f*)
  show *?case* **proof**(*cases f*)
    case (*Inr g*)
    with *2* **show** *?thesis* **unfolding** *Inr*
      by (*simp add*: *nth-map-conv*)

```
  next
    case (Inl α)
    with 2 show ?thesis unfolding Inl
      by (metis nth-map-conv source.simps(3))
  qed
next
  case (3 As Bs α)
  then have l:length (zip As Bs) = length As
    by simp
  with 3 have IH:∀ i < length As. source (As!i) = source (Bs!i)
    by (metis (mono-tags, lifting) case-prod-conv nth-mem nth-zip)
  have src:source ((to-pterm (lhs α)) · ⟨Bs⟩α) = (lhs α) · ⟨map source Bs⟩α
    by (simp add: source-apply-subst)
  from 3(1) l IH show ?case unfolding src source.simps
    by (metis nth-map-conv)
next
  case (4 As Bs α)
  then have l:length (zip As Bs) = length As
    by simp
  with 4 have IH:∀ i < length As. source (As!i) = source (Bs!i)
    by (metis (mono-tags, lifting) case-prod-conv nth-mem nth-zip)
  have src:source ((to-pterm (lhs α)) · ⟨As⟩α) = (lhs α) · ⟨map source As⟩α
    by (simp add: source-apply-subst)
  from 4(1) l IH show ?case unfolding src source.simps
    by (metis nth-map-conv)
qed simp
```

If two proof terms are orthogonal then residual and join are well-defined.

```
lemma orth-imp-residual-defined:
  assumes varcond:⋀l r. (l, r) ∈ R ⟹ is-Fun l ⋀l r. (l, r) ∈ S ⟹ is-Fun l
    and A ⊥p B
    and A ∈ wf-pterm R and B ∈ wf-pterm S
  shows A re B ≠ None
  using assms(3−) proof(induct)
  case (2 As Bs f)
  from 2(3) have wellAs:∀ a ∈ set As. a ∈ wf-pterm R
    by blast
  from 2(4) have wellBs:∀ b ∈ set Bs. b ∈ wf-pterm S
    by blast
  from 2(1,2) wellAs wellBs  have c:∀ i < length As. (∃ C. As!i re Bs!i = Some
C)
    by auto
  from 2(1) have l:length As = length (map2 (re) As Bs)
    by simp
  from 2(1) have ∀ i < length As. As!i re Bs!i = (map2 (re) As Bs)!i
    by simp
 with c obtain Cs where ∀ i < length As. As!i re Bs!i = Some (Cs!i) and length
Cs = length As
    using exists-some-list l by (metis (no-types, lifting))
```

with *2* **have** ∗:*those* (*map2* (*re*) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
show *?case* **proof**(*cases f*)
case (*Inr g*)
show *?thesis* **unfolding** *Inr residual.simps 2*(*1*) ∗ **by** *simp*
next
case (*Inl α*)
show *?thesis* **unfolding** *Inl residual.simps 2*(*1*) ∗ **by** *simp*
qed
next
case (*3 As Bs α*)
from *3*(*3*) *varcond* **obtain** *g ts* **where** *g*:*lhs α* = *Fun g ts*
by (*metis Inl-inject is-Fun-Fun-conv sum.simps*(*4*) *term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.cases*)
then have ∗:*to-pterm* (*lhs α*) · ⟨*Bs*⟩$_\alpha$ = *Pfun g* (*map* (λ*t*. *t* · ⟨*Bs*⟩$_\alpha$) (*map to-pterm ts*))
by *simp*
from *3*(*3*) **have** *l1*:*length As* = *length* (*var-rule α*)
using *wf-pterm.simps* **by** *fastforce*
from *3*(*3*) **have** *wellAs*:∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R*
by *blast*
from *3*(*1*,*4*) *l1* **have** *wellBs*:∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm S*
by (*simp add*: *lhs-subst-args-wf-pterm*)
from *3*(*1*) **have** *l2*:*length* (*zip As Bs*) = *length As*
by *simp*
with *3*(*1*,*2*) *wellAs wellBs* **have** ∀ *i* < *length As*. *As* ! *i re Bs* ! *i* ≠ *None*
by (*metis* (*mono-tags*, *lifting*) *case-prod-conv nth-mem nth-zip*)
then have *c*:∀ *i* < *length As*. (∃ *C*. *As*!*i re Bs*!*i* = *Some C*)
by *blast*
from *3*(*1*) **have** ∀ *i* < *length As*. *As*!*i re Bs*!*i* = (*map2* (*re*) *As Bs*)!*i*
by *simp*
with *c* **obtain** *Cs* **where** ∀ *i* < *length As*. *As*!*i re Bs*!*i* = *Some* (*Cs*!*i*) **and** *length Cs* = *length As*
using *exists-some-list l2* **by** (*metis* (*no-types*, *lifting*) *length-map*)
with *3* **have** *cs*:*those* (*map2* (*re*) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
have *bs*:*match* (*to-pterm* (*lhs α*) · ⟨*Bs*⟩$_\alpha$) (*to-pterm* (*lhs α*)) = *Some* (⟨*Bs*⟩$_\alpha$)
using *lhs-subst-trivial* **by** *blast*
then have (*map* (⟨*Bs*⟩$_\alpha$) (*var-rule α*)) = *Bs*
using *3*(*1*) *l1 apply-lhs-subst-var-rule* **by** *force*
then show *?case* **using** *residual.simps*(*5*) **using** *bs cs g* **unfolding** ∗
by *simp*
next
case (*4 As Bs α*)
from *4*(*4*) *varcond* **obtain** *g ts* **where** *g*:*lhs α* = *Fun g ts*
by (*metis Inl-inject is-Fun-Fun-conv sum.simps*(*4*) *term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.cases*)
then have ∗:*to-pterm* (*lhs α*) · ⟨*As*⟩$_\alpha$ = *Pfun g* (*map* (λ*t*. *t* · ⟨*As*⟩$_\alpha$) (*map to-pterm ts*))

by *simp*
from *4*(*4*) **have** *l1*:*length Bs* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
from *4*(*1*,*3*) *l1* **have** *wellAs*:$\forall$ *a* $\in$ *set As*. *a* $\in$ *wf-pterm R*
by (*simp add*: *lhs-subst-args-wf-pterm*)
from *4*(*4*) **have** *wellBs*:$\forall$ *b* $\in$ *set Bs*. *b* $\in$ *wf-pterm S*
by *blast*
from *4*(*1*) **have** *l2*:*length* (*zip As Bs*) = *length As*
by *simp*
with *4*(*1*,*2*) *wellAs wellBs* **have** $\forall$ *i* < *length As*. *As* ! *i re Bs* ! *i* $\neq$ *None*
by (*metis* (*mono-tags*, *lifting*) *case-prod-conv nth-mem nth-zip*)
then have *c*:$\forall$ *i* < *length As*. ($\exists$ *C*. *As*!*i re Bs*!*i* = *Some C*)
by *blast*
from *4*(*1*) **have** $\forall$ *i* < *length As*. *As*!*i re Bs*!*i* = (*map2* (*re*) *As Bs*)!*i*
by *simp*
with *c* **obtain** *Cs* **where** $\forall$ *i* < *length As*. *As*!*i re Bs*!*i* = *Some* (*Cs*!*i*) **and** *length Cs* = *length As*
using *exists-some-list l2* **by** (*metis* (*no-types*, *lifting*) *length-map*)
with *4* **have** *cs*:*those* (*map2* (*re*) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
have *bs*:*match* (*to-pterm* (*lhs* α) $\cdot$ $\langle As \rangle_\alpha$) (*to-pterm* (*lhs* α)) = *Some* ($\langle As \rangle_\alpha$)
using *lhs-subst-trivial* **by** *blast*
have (*map* ($\langle As \rangle_\alpha$) (*var-rule* α)) = *As*
using *4*(*1*) *l1 apply-lhs-subst-var-rule* **by** *force*
then show *?case* **using** *residual.simps*(*7*) **using** *bs cs g* **unfolding** $*$
by *simp*
qed *simp*

lemma *orth-imp-join-defined*:
assumes *varcond*:$\bigwedge$*l r*. (*l*, *r*) $\in$ *R* $\Longrightarrow$ *is-Fun l*
and *A* $\perp_p$ *B*
and *A* $\in$ *wf-pterm R* **and** *B* $\in$ *wf-pterm R*
shows *A* $\sqcup$ *B* $\neq$ *None*
using *assms*(*2*$-$) **proof**(*induct*)
case (*2 As Bs f*)
from *2*(*3*) **have** *wellAs*:$\forall$ *a* $\in$ *set As*. *a* $\in$ *wf-pterm R*
by *blast*
from *2*(*4*) **have** *wellBs*:$\forall$ *b* $\in$ *set Bs*. *b* $\in$ *wf-pterm R*
by *blast*
from *2*(*1*,*2*) *wellAs wellBs* **have** *c*:$\forall$ *i* < *length As*. ($\exists$ *C*. *As*!*i* $\sqcup$ *Bs*!*i* = *Some C*)
by *auto*
from *2*(*1*) **have** *l*:*length As* = *length* (*map2* ($\sqcup$) *As Bs*)
by *simp*
from *2*(*1*) **have** $\forall$ *i* < *length As*. *As*!*i* $\sqcup$ *Bs*!*i* = (*map2* ($\sqcup$) *As Bs*)!*i*
by *simp*
with *c* **obtain** *Cs* **where** $\forall$ *i* < *length As*. *As*!*i* $\sqcup$ *Bs*!*i* = *Some* (*Cs*!*i*) **and** *length Cs* = *length As*

using *exists-some-list l* **by** (*metis* (*no-types*, *lifting*))
with *2* **have** ∗:*those* (*map2* (⊔) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
show *?case* **proof**(*cases f*)
case (*Inr g*)
show *?thesis* **unfolding** *Inr join.simps 2*(*1*) ∗ **by** *simp*
next
case (*Inl α*)
show *?thesis* **unfolding** *Inl join.simps 2*(*1*) ∗ **by** *simp*
qed
next
case (*3 As Bs α*)
from *3*(*3*) *varcond* **obtain** *g ts* **where** *g*:*lhs α* = *Fun g ts*
by (*metis Inl-inject is-Fun-Fun-conv sum.simps*(*4*) *term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.cases*)
then have ∗:*to-pterm* (*lhs α*) · ⟨*Bs*⟩$_\alpha$ = *Pfun g* (*map* (λ*t*. *t* · ⟨*Bs*⟩$_\alpha$) (*map to-pterm ts*))
by *simp*
from *3*(*3*) **have** *l1*:*length As* = *length* (*var-rule α*)
using *wf-pterm.simps* **by** *fastforce*
from *3*(*3*) **have** *wellAs*:∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R*
by *blast*
from *3*(*1*,*4*) *l1* **have** *wellBs*:∀ *b* ∈ *set Bs*. *b* ∈ *wf-pterm R*
by (*simp add*: *lhs-subst-args-wf-pterm*)
from *3*(*1*) **have** *l2*:*length* (*zip As Bs*) = *length As*
by *simp*
with *3*(*1*,*2*) *wellAs wellBs* **have** ∀ *i* < *length As*. *As* ! *i* ⊔ *Bs* ! *i* ≠ *None*
by (*metis* (*mono-tags*, *lifting*) *case-prod-conv nth-mem nth-zip*)
then have *c*:∀ *i* < *length As*. (∃ *C*. *As*!*i* ⊔ *Bs*!*i* = *Some C*)
by *blast*
from *3*(*1*) **have** ∀ *i* < *length As*. *As*!*i* ⊔ *Bs*!*i* = (*map2* (⊔) *As Bs*)!*i*
by *simp*
with *c* **obtain** *Cs* **where** ∀ *i* < *length As*. *As*!*i* ⊔ *Bs*!*i* = *Some* (*Cs*!*i*) **and** *length Cs* = *length As*
using *exists-some-list l2* **by** (*metis* (*no-types*, *lifting*) *length-map*)
with *3* **have** *cs*:*those* (*map2* (⊔) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
have *bs*:*match* (*to-pterm* (*lhs α*) · ⟨*Bs*⟩$_\alpha$) (*to-pterm* (*lhs α*)) = *Some* (⟨*Bs*⟩$_\alpha$)
using *lhs-subst-trivial* **by** *blast*
then have (*map* (⟨*Bs*⟩$_\alpha$) (*var-rule α*)) = *Bs*
using *3*(*1*) *l1 apply-lhs-subst-var-rule* **by** *force*
then show *?case* **using** *residual.simps*(*5*) **using** *bs cs g* **unfolding** ∗
by *simp*
next
case (*4 As Bs α*)
from *4*(*4*) *varcond* **obtain** *g ts* **where** *g*:*lhs α* = *Fun g ts*
by (*metis Inl-inject is-Fun-Fun-conv sum.simps*(*4*) *term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.cases*)
then have ∗:*to-pterm* (*lhs α*) · ⟨*As*⟩$_\alpha$ = *Pfun g* (*map* (λ*t*. *t* · ⟨*As*⟩$_\alpha$) (*map*

to-pterm ts))
by *simp*
from *4*(*4*) **have** *l1*:*length Bs* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
from *4*(*1*,*3*) *l1* **have** *wellAs*:$\forall$ *a* $\in$ *set As*. *a* $\in$ *wf-pterm R*
by (*simp add*: *lhs-subst-args-wf-pterm*)
from *4*(*4*) **have** *wellBs*:$\forall$ *b* $\in$ *set Bs*. *b* $\in$ *wf-pterm R*
by *blast*
from *4*(*1*) **have** *l2*:*length* (*zip As Bs*) = *length As*
by *simp*
with *4*(*1*,*2*) *wellAs wellBs* **have** $\forall$ *i* < *length As*. *As* ! *i* $\sqcup$ *Bs* ! *i* $\neq$ *None*
by (*metis* (*mono-tags*, *lifting*) *case-prod-conv nth-mem nth-zip*)
then have *c*:$\forall$ *i* < *length As*. ($\exists$ *C*. *As*!*i* $\sqcup$ *Bs*!*i* = *Some C*)
by *blast*
from *4*(*1*) **have** $\forall$ *i* < *length As*. *As*!*i* $\sqcup$ *Bs*!*i* = (*map2* ($\sqcup$) *As Bs*)!*i*
by *simp*
with *c* **obtain** *Cs* **where** $\forall$ *i* < *length As*. *As*!*i* $\sqcup$ *Bs*!*i* = *Some* (*Cs*!*i*) **and** *length Cs* = *length As*
using *exists-some-list l2* **by** (*metis* (*no-types*, *lifting*) *length-map*)
with *4* **have** *cs*:*those* (*map2* ($\sqcup$) *As Bs*) = *Some Cs*
by (*simp add*: *those-some*)
have *bs*:*match* (*to-pterm* (*lhs* α) $\cdot$ $\langle$*As*$\rangle_\alpha$) (*to-pterm* (*lhs* α)) = *Some* ($\langle$*As*$\rangle_\alpha$)
using *lhs-subst-trivial* **by** *blast*
have (*map* ($\langle$*As*$\rangle_\alpha$) (*var-rule* α)) = *As*
using *4*(*1*) *l1 apply-lhs-subst-var-rule* **by** *force*
then show *?case* **using** *residual.simps*(*7*) **using** *bs cs g* **unfolding** $*$
by *simp*
qed *simp*

context *no-var-lhs*
begin
lemma *orth-imp-residual-defined*:
assumes *A* $\perp_p$ *B* **and** *A* $\in$ *wf-pterm R* **and** *B* $\in$ *wf-pterm R*
shows *A re B* $\neq$ *None*
using *orth-imp-residual-defined assms no-var-lhs* **by** *fastforce*

lemma *orth-imp-join-defined*:
assumes *A* $\perp_p$ *B* **and** *A* $\in$ *wf-pterm R* **and** *B* $\in$ *wf-pterm R*
shows *A* $\sqcup$ *B* $\neq$ *None*
using *orth-imp-join-defined assms no-var-lhs* **by** *fastforce*

lemma *orthogonal-ctxt*:
assumes *C*$\langle$*A*$\rangle$ $\perp_p$ *C*$\langle$*B*$\rangle$ *C* $\in$ *wf-pterm-ctxt R*
shows *A* $\perp_p$ *B*
using *assms* **proof**(*induct C*)
case (*Cfun f ss1 C ss2*)
from *Cfun*(*2*) **have** $\forall$ *i*<*length* (*ss1* @ *C*$\langle$*A*$\rangle$ # *ss2*). (*ss1* @ *C*$\langle$*A*$\rangle$ # *ss2*) ! *i* $\perp_p$ (*ss1* @ *C*$\langle$*B*$\rangle$ # *ss2*) ! *i*
unfolding *intp-actxt.simps* **using** *orthogonal.simps* **by** (*smt* (*verit*) *is-Prule.simps*(*1*)

is-Prule.simps(*3*) *term.distinct*(*1*) *term.sel*(*4*))
then have $C\langle A\rangle \perp_p C\langle B\rangle$
by (*metis append.right-neutral length-append length-greater-0-conv length-nth-simps*(*1*) *list.discI nat-add-left-cancel-less nth-append-length*)
with *Cfun*(*1*,*3*) **show** *?case* **by** *auto*
next
case (*Crule* α *ss1 C ss2*)
from *Crule*(*3*) **obtain** *f ts* **where** *lhs* α = *Fun f ts*
using *no-var-lhs* **by** (*smt* (*verit*, *del-insts*) *Inl-inject Inr-not-Inl case-prodD actxt.distinct*(*1*) *actxt.inject term.collapse*(*2*) *wf-pterm-ctxt.simps*)
with *Crule*(*2*) **have** $\forall i{<}$*length* (*ss1* @ $C\langle A\rangle$ # *ss2*). (*ss1* @ $C\langle A\rangle$ # *ss2*) ! *i* $\perp_p$ (*ss1* @ $C\langle B\rangle$ # *ss2*) ! *i*
unfolding *intp-actxt.simps* **using** *orthogonal.simps*
by (*smt* (*verit*, *ccfv-threshold*) *Inl-inject Inr-not-Inl eval-term.simps*(*2*) *term.distinct*(*1*) *term.inject*(*2*) *to-pterm.simps*(*2*))
then have $C\langle A\rangle \perp_p C\langle B\rangle$
by (*metis intp-actxt.simps*(*2*) *hole-pos.simps*(*2*) *hole-pos-poss nth-append-length poss-Cons-poss term.sel*(*4*))
with *Crule*(*1*,*3*) **show** *?case* **by** *auto*
qed *simp*

end

context *left-lin-no-var-lhs*
begin

lemma *orthogonal-subst*:
assumes $A \cdot \sigma \perp_p B \cdot \sigma$ *source A* = *source B*
and $A \in$ *wf-pterm R B* $\in$ *wf-pterm R*
shows $A \perp_p B$
using *assms*(*3*,*4*,*1*,*2*) **proof**(*induct A arbitrary*:*B rule*:*subterm-induct*)
case (*subterm A*)
show *?case* **proof**(*cases A*)
case (*Var x*)
with *subterm no-var-lhs* **have** *B* = *Var x*
by (*metis Inl-inject Inr-not-Inl case-prodD co-initial-Var is-VarI term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
then show *?thesis*
unfolding *Var* **by** (*simp add*: *orthogonal.intros*(*1*))
next
case (*Pfun f As*)
with *subterm*(*5*) **show** *?thesis* **proof**(*cases B*)
case (*Pfun g Bs*)
from *subterm*(*5*) **have** *f*:*f* = *g*
unfolding ‹*A* = *Pfun f As*› *Pfun* **by** *simp*
from *subterm*(*5*) **have** *l*:*length As* = *length Bs*
unfolding ‹*A* = *Pfun f As*› *Pfun* **using** *map-eq-imp-length-eq* **by** *auto*
{fix *i* **assume** *i*:*i* < *length As*
with *subterm*(*4*) **have** *As*!*i* $\cdot \sigma \perp_p$ *Bs*!*i* $\cdot \sigma$

```
      unfolding ‹A = Pfun f As› Pfun eval-term.simps f
       by (smt (verit) is-Prule.simps(1) is-Prule.simps(3) length-map nth-map
orthogonal.simps term.distinct(1) term.sel(4))
     moreover from i subterm(5) have source (As!i) = source (Bs!i)
         unfolding ‹A = Pfun f As› Pfun eval-term.simps f by (simp add:
map-equality-iff)
     moreover from i l subterm(2,3) have As!i ∈ wf-pterm R Bs!i ∈ wf-pterm
R
      unfolding ‹A = Pfun f As› Pfun by auto
     moreover from i have As!i ◁ A
      unfolding ‹A = Pfun f As› by simp
     ultimately have As!i ⊥p Bs!i
      using subterm(1) by simp
   }
   with l show ?thesis
    unfolding f ‹A = Pfun f As› Pfun by (simp add: orthogonal.intros(2))
  next
   case (Prule β Bs)
   with subterm(3) have lin:linear-term (lhs β)
    using left-lin left-linear-trs-def wf-pterm.cases by fastforce
   from subterm(3) obtain g ts where lhs:lhs β = Fun g ts
    unfolding Prule using no-var-lhs by (metis Inl-inject case-prodD is-FunE
is-Prule.simps(1) is-Prule.simps(3) term.distinct(1) term.inject(2) wf-pterm.simps)

   with subterm(4) obtain τ1 where A · σ = to-pterm (lhs β) · τ1
     unfolding Prule Pfun eval-term.simps using orthogonal.simps by (smt
(verit, ccfv-SIG) Inl-inject Inr-not-Inl term.inject(2))
   with subterm(4,5) obtain τ where τ2:A = to-pterm (lhs β) · τ
    unfolding Prule Pfun source.simps using simple-pterm-match lin by (metis
matches-iff source.simps(2))
   let ?As=map τ (var-rule β)
   have l:length Bs = length (var-rule β)
    using Prule subterm.prems(2) wf-pterm.simps by fastforce
   from τ2 have A:A = to-pterm (lhs β) · ⟨?As⟩β
    by (metis lhs-subst-var-rule set-vars-term-list subsetI vars-to-pterm)
   {fix i assume i:i < length Bs
    have subt:?As!i ◁ A
     using i l by (metis (no-types, lifting) τ2 comp-apply lhs nth-map nth-mem
set-remdups set-rev set-vars-term-list subst-image-subterm to-pterm.simps(2) vars-to-pterm)
    have wf:?As!i ∈ wf-pterm R
     using i l by (metis A length-map lhs-subst-args-wf-pterm nth-mem sub-
term.prems(1))
    have l':length (var-rule β) = length ?As
     by simp
    from subterm(4) A Prule lhs have orth:?As!i · σ ⊥p Bs!i · σ proof(cases)
     case (4 As' Bs' β')
     then have Bs':Bs' = map (λs. s · σ) Bs and β':β' = β
      unfolding Prule by simp-all
     have As':As' = map (λs. s · σ) ?As proof−
```

have l'':*length* As' = *length* *?As* **using** *4*(*3*) l **unfolding** Bs' *length-map*
by *simp*
{fix j **assume** j:$j < length\ As'$ **and** *neq*:$As'!j \neq map\ (\lambda s.\ s \cdot \sigma)\ ?As\ !\ j$
let *?x*=*var-rule* $\beta!j$
from j **have** x:$?x \in vars\text{-}term\ (lhs\ \beta)$
by (*metis comp-apply* l' l'' *nth-mem set-remdups set-rev set-vars-term-list*)

then obtain p **where** p:$p \in poss\ (lhs\ \beta)\ lhs\ \beta\ |\text{-}p = Var\ ?x$
by (*meson vars-term-poss-subt-at*)
from j **have** *1*:$(\langle As' \rangle_{\beta'})\ ?x = As'!j$
using β' l'' *lhs-subst-var-i* **by** *force*
from j **have** *2*:$(\langle map\ (\lambda s.\ s \cdot \sigma)\ ?As \rangle_{\beta})\ ?x = map\ (\lambda s.\ s \cdot \sigma)\ ?As\ !\ j$
using *lhs-subst-var-i* **by** (*metis* l'' *length-map*)
then have *False* **using** *4*(*1*) *1 2 p* **unfolding** *A eval-lhs-subst*[*OF* l'] β'
by (*smt* (*verit*, *del-insts*) *x neq set-vars-term-list term-subst-eq-rev vars-to-pterm*)
}
then show *?thesis*
using l'' **by** (*metis* (*mono-tags*, *lifting*) *map-nth-eq-conv nth-map*)
qed
have i':$i < length\ (zip\ (map\ (\lambda a.\ a \cdot \sigma)\ (map\ \tau\ (var\text{-}rule\ \beta)))\ (map\ (\lambda b.\ b \cdot \sigma)\ Bs))$
using i l **by** *simp*
from *4*(*4*) **have** $map\ (\lambda a.\ a \cdot \sigma)\ (map\ \tau\ (var\text{-}rule\ \beta))\ !\ i \perp_p map\ (\lambda b.\ b \cdot \sigma)\ Bs\ !i$
unfolding As' Bs' **using** i' **by** (*metis case-prodD i l length-map nth-mem nth-zip*)
then show *?thesis*
using i l **by** *auto*
qed *simp-all*
have *co-init*:*source* (*?As*!i) = *source* ($Bs!i$) **proof**(*rule ccontr*)
assume *neq*:*source* (*?As*!i) ≠ *source* ($Bs!i$)
let *?x*=*var-rule* $\beta!i$
from i l **have** x:$?x \in vars\text{-}term\ (lhs\ \beta)$
by (*metis comp-apply nth-mem set-remdups set-rev set-vars-term-list*)
then obtain p **where** p:$p \in poss\ (lhs\ \beta)\ lhs\ \beta\ |\text{-}p = Var\ ?x$
by (*meson vars-term-poss-subt-at*)
from i **have** *1*:$(\langle map\ source\ ?As \rangle_{\beta})\ ?x = source\ (?As!i)$
using l *lhs-subst-var-i* **by** (*metis length-map nth-map*)
from i **have** *2*:$(\langle map\ source\ Bs \rangle_{\beta})\ ?x = source\ (Bs!i)$
using l *lhs-subst-var-i* **by** (*metis length-map nth-map*)
from *subterm*(*5*) **show** *False* **using** *neq 1 2 p*
unfolding *Prule A source.simps source-apply-subst*[*OF to-pterm-wf-pterm*[*of lhs* β]] *source-to-pterm*
using *term-subst-eq-rev x* **by** *fastforce*
qed
from *subterm*(*1*) *subt wf orth co-init* **have** *?As*!$i \perp_p Bs!i$
using i *subterm*(*3*) **unfolding** *Prule* **by** (*meson fun-well-arg nth-mem*)

```
    }
    then show ?thesis unfolding A Prule
    by (smt (verit, best) case-prodI2 in-set-idx l length-map length-zip min-less-iff-conj
nth-zip orthogonal.intros(4) prod.sel(1) snd-conv)
  qed simp
 next
  case (Prule α As)
  with subterm(2) have lin:linear-term (lhs α)
   using left-lin left-linear-trs-def wf-pterm.cases by fastforce
  from subterm(2) obtain f ts where lhs:lhs α = Fun f ts
     by (metis Inr-not-Inl Prule case-prodD is-FunE no-var-lhs sum.inject(1)
term.distinct(1) term.inject(2) wf-pterm.simps)
  with subterm(5) show ?thesis proof(cases B)
   case (Var x)
   then show ?thesis
    using source-orthogonal subterm.prems(4) by fastforce
  next
   case (Pfun g Bs)
   with subterm(4) obtain τ1 where B · σ = to-pterm (lhs α) · τ1
     unfolding Prule Pfun eval-term.simps using orthogonal.simps by (smt
(verit, ccfv-SIG) Inl-inject Inr-not-Inl term.inject(2))
   with subterm(4,5) obtain τ where τ2:B = to-pterm (lhs α) · τ
    unfolding Prule Pfun source.simps using simple-pterm-match lin by (metis
matches-iff source.simps(2))
   let ?Bs=map τ (var-rule α)
   have l:length As = length (var-rule α)
    using Prule subterm.prems(1) wf-pterm.simps by fastforce
   from τ2 have B:B = to-pterm (lhs α) · ⟨?Bs⟩α
    by (metis lhs-subst-var-rule set-vars-term-list subsetI vars-to-pterm)
   have l':length (var-rule α) = length ?Bs
    by simp
   {fix i assume i:i < length As
    from subterm(4) B Prule lhs have orth:As!i · σ ⊥p ?Bs!i · σ proof(cases)
     case (3 As' Bs' α')
     then have As':As' = map (λs. s · σ) As and α':α' = α
      unfolding Prule by simp-all
     have Bs':Bs' = map (λs. s · σ) ?Bs proof−
      have l'':length Bs' = length ?Bs using 3(3) l unfolding As' length-map
by simp
       {fix j assume j:j < length Bs' and neq:Bs'!j ≠ map (λs. s · σ) ?Bs ! j
        let ?x=var-rule α!j
        from j have x:?x ∈ vars-term (lhs α)
          by (metis comp-apply l'' length-map nth-mem set-remdups set-rev
set-vars-term-list)
        then obtain p where p:p ∈ poss (lhs α) lhs α |-p = Var ?x
         by (meson vars-term-poss-subt-at)
        from j have 1:(⟨Bs'⟩α') ?x = Bs'!j
         using α' l'' lhs-subst-var-i by force
        from j have 2:(⟨map (λs. s · σ) ?Bs⟩α) ?x = map (λs. s · σ) ?Bs ! j
```

```
            using lhs-subst-var-i by (metis l'' length-map)
           then have False using 3(1) 1 2 p unfolding B eval-lhs-subst[OF l']
α'
            by (smt (verit, ccfv-SIG) 3(2) B α' eval-lhs-subst l' map-eq-conv neq
set-vars-term-list term-subst-eq-rev vars-to-pterm x)
          }
          then show ?thesis
           using l'' by (metis (mono-tags, lifting) map-nth-eq-conv nth-map)
         qed
          have i':i < length (zip (map (λb. b · σ) As) (map (λa. a · σ) (map τ
(var-rule α))))
          using i l by simp
        from 3(4) have map (λb. b · σ) As ! i ⊥p map (λa. a · σ) (map τ (var-rule
α)) ! i
         unfolding As' Bs' using i' by (metis case-prodD i l length-map nth-mem
nth-zip)
        then show ?thesis
          using i l by auto
       qed simp-all
       have co-init:source (As!i) = source (?Bs!i) proof(rule ccontr)
        assume neq:source (As!i) ≠ source (?Bs!i)
        let ?x=var-rule α!i
        from i l have x:?x ∈ vars-term (lhs α)
          by (metis comp-apply nth-mem set-remdups set-rev set-vars-term-list)
        then obtain p where p:p ∈ poss (lhs α) lhs α |-p = Var ?x
          by (meson vars-term-poss-subt-at)
        from i have 1:(⟨map source ?Bs⟩α) ?x = source (?Bs!i)
          using l lhs-subst-var-i by (metis length-map nth-map)
        from i have 2:(⟨map source As⟩α) ?x = source (As!i)
          using l lhs-subst-var-i by (metis length-map nth-map)
        from subterm(5) show False using neq 1 2 p
       unfolding Prule B source.simps source-apply-subst[OF to-pterm-wf-pterm[of
lhs α]] source-to-pterm
          using term-subst-eq-rev x by fastforce
       qed
       from subterm(1,2,3) co-init have As!i ⊥p ?Bs!i
      using i l' orth unfolding Prule by (metis B fun-well-arg l lhs-subst-args-wf-pterm
nth-mem orth supt.arg)
     }
     then show ?thesis unfolding B Prule
      by (smt (verit, best) case-prodI2 fst-conv in-set-zip l l' orthogonal.intros(3)
snd-conv)
   next
     case (Prule β Bs)
     from subterm(4) have α:α = β
      unfolding Prule ‹A = Prule α As› eval-term.simps using orthogonal.simps
          by (smt (verit) Inl-inject Prule eval-term.simps(2) is-Prule.simps(1)
is-Prule.simps(3) lhs no-var-lhs.lhs-is-Fun
                no-var-lhs-axioms subterm.prems(2) term.collapse(2) term.sel(2)
```

to-pterm.simps(*2*))
from *subterm*(*2*,*3*) **have** *l*:*length As* = *length Bs*
unfolding ‹*A* = *Prule* α *As*› *Prule* **using** α *length-args-well-Prule* **by** *blast*
{fix *i* **assume** *i*:*i* < *length As*
with *subterm*(*4*) **have** *As*!*i* · σ ⊥*p* *Bs*!*i* · σ
unfolding ‹*A* = *Prule* α *As*› *Prule eval-term.simps* α **by** (*smt* (*verit*, *ccfv-threshold*) *Inl-inject*
Inr-not-Inl α *eval-term.simps*(*2*) *length-map lhs nth-map orthogonal.simps term.distinct*(*1*) *term.inject*(*2*) *to-pterm.simps*(*2*))
moreover from *i subterm*(*5*) **have** *source* (*As*!*i*) = *source* (*Bs*!*i*)
using *Prule* α ‹*A* = *Prule* α *As*› *co-init-prule subterm.prems*(*1*) *subterm.prems*(*2*) **by** *blast*
moreover from *i l subterm*(*2*,*3*) **have** *As*!*i* ∈ *wf-pterm R Bs*!*i* ∈ *wf-pterm R*
unfolding *Prule* ‹*A* = *Prule* α *As*› **by** *auto*
moreover from *i* **have** *As*!*i* ◁ *A*
unfolding ‹*A* = *Prule* α *As*› **by** *simp*
ultimately have *As*!*i* ⊥*p* *Bs*!*i*
using *subterm*(*1*) **by** *simp*
}
with *l* **show** *?thesis*
unfolding α ‹*A* = *Prule* α *As*› *Prule* **by** (*simp add*: *orthogonal.intros*(*2*))
qed
qed
qed

end

end

5 Labels and Overlaps

theory *Labels-and-Overlaps*
imports
Orthogonal-PT
Well-Quasi-Orders.Almost-Full-Relations
begin

5.1 Labeled Proof Terms

The idea is to label function symbols in the source of a proof term that are affected by a rule symbol α with α and the distance from the root to α. Therefore, a label is a pair consisting of a rule symbol and a natural number, or it can be *None*. A labeled term is a term, where each function symbol additionally has a label associated with it.

type-synonym
('*f*, '*v*) *label* = (('*f*, '*v*) *prule* × *nat*) *option*
type-synonym

```
  ('f, 'v) term-lab = ('f × ('f, 'v) label, 'v) term

fun label-term :: ('f, 'v) prule ⇒ nat ⇒ ('f, 'v) term ⇒ ('f, 'v) term-lab
  where
  label-term α i (Var x) = Var x
| label-term α i (Fun f ts) = Fun (f, Some (α, i)) (map (label-term α (i+1)) ts)

abbreviation labeled-lhs :: ('f, 'v) prule ⇒ ('f, 'v) term-lab
  where labeled-lhs α ≡ label-term α 0 (lhs α)

fun labeled-source :: ('f, 'v) pterm ⇒ ('f, 'v) term-lab
  where
  labeled-source (Var x) = Var x
| labeled-source (Pfun f As) = Fun (f, None) (map labeled-source As)
| labeled-source (Prule α As) = (labeled-lhs α) · ⟨map labeled-source As⟩α

fun term-lab-to-term :: ('f, 'v) term-lab ⇒ ('f, 'v) term
  where
  term-lab-to-term (Var x) = Var x
| term-lab-to-term (Fun f ts) = Fun (fst f) (map term-lab-to-term ts)

fun term-to-term-lab :: ('f, 'v) term ⇒ ('f, 'v) term-lab
  where
  term-to-term-lab (Var x) = Var x
| term-to-term-lab (Fun f ts) = Fun (f, None) (map term-to-term-lab ts)

fun get-label :: ('f, 'v) term-lab ⇒ ('f, 'v) label
  where
  get-label (Var x) = None
| get-label (Fun f ts) = snd f

fun labelposs :: ('f, 'v) term-lab ⇒ pos set
where
   labelposs (Var x) = {}
 | labelposs (Fun (f, None) ts) = (⋃ i<length ts. {i # p | p. p ∈ labelposs (ts ! i)})
 | labelposs (Fun (f, Some l) ts) = {[]} ∪ (⋃ i<length ts. {i # p | p. p ∈ labelposs
(ts ! i)})

abbreviation possL :: ('f, 'v) pterm ⇒ pos set
  where possL A ≡ labelposs (labeled-source A)

lemma labelposs-term-to-term-lab: labelposs (term-to-term-lab t) = {}
  by(induct t) simp-all

lemma term-lab-to-term-lab[simp]: term-lab-to-term (term-to-term-lab t) = t
proof(induct t)
  case (Fun f ts)
  then show ?case
    unfolding term-lab-to-term.simps term-to-term-lab.simps fst-conv by (simp
```

```
add: map-nth-eq-conv)
qed simp

lemma term-lab-to-term-subt-at:
  assumes p ∈ poss t
  shows term-lab-to-term t |-p = term-lab-to-term (t|-p)
  using assms proof(induct p arbitrary:t)
  case (Cons i p)
  from args-poss[OF Cons(2)] obtain f ts where f:t = Fun f ts and
    p:p ∈ poss (ts ! i) and i:i < length ts by blast
  from Cons(1)[OF p] i show ?case
    unfolding f term-lab-to-term.simps by simp
qed simp

lemma vars-term-labeled-lhs: vars-term (label-term α i t) = vars-term t
  by (induct t arbitrary:i) simp-all

lemma vars-term-list-labeled-lhs: vars-term-list (label-term α i t) = vars-term-list
t
proof (induct t arbitrary:i)
  case (Fun f ts)
  show ?case unfolding label-term.simps vars-term-list.simps using Fun
    by (metis (mono-tags, lifting) length-map map-nth-eq-conv nth-mem)
qed (simp add: vars-term-list.simps(1))

lemma var-poss-list-labeled-lhs: var-poss-list (label-term α i t) = var-poss-list t
proof (induct t arbitrary:i)
  case (Fun f ts)
  then have ts:map (var-poss-list ∘ label-term α (i + 1)) ts = map var-poss-list
ts
    by auto
  then show ?case
    unfolding label-term.simps var-poss-list.simps map-map ts by simp
qed (simp add: poss-list.simps(1))

lemma var-labeled-lhs[simp]: vars-distinct (label-term α i t) = vars-distinct t
  by (simp add: vars-term-list-labeled-lhs)

lemma labelposs-subt-at:
  assumes q ∈ poss t p ∈ labelposs (t|-q)
  shows q@p ∈ labelposs t
  using assms proof(induct t arbitrary:q)
  case (Fun f ts)
  then show ?case proof(cases q)
    case (Cons i q')
    with Fun(2) have i:i < length ts and q':q' ∈ poss (ts!i)
      by simp+
    with Fun(3) have p ∈ labelposs ((ts!i)|-q')
      unfolding Cons by simp
```

```
    with Fun(1) i q' have IH:q'@p ∈ labelposs (ts!i)
      using nth-mem by blast
    obtain f' lab where f:f = (f', lab)
      by fastforce
    then show ?thesis proof(cases lab)
      case None
      show ?thesis
        unfolding f Cons None labelposs.simps using i IH by simp
    next
      case (Some a)
      then show ?thesis
        unfolding f Cons Some labelposs.simps using i IH by simp
    qed
  qed simp
qed simp

lemma var-label-term:
  assumes p ∈ poss t and t|-p = Var x
  shows label-term α n t |-p = Var x
  using assms proof(induct t arbitrary:p n)
  case (Fun f ts)
  then obtain i p' where p':i < length ts p = i#p' p' ∈ poss (ts!i)
    by auto
  then show ?case
    unfolding label-term.simps p'(2) subt-at.simps using Fun(1,3) p'(2) by force
qed simp

lemma get-label-label-term:
  assumes p ∈ fun-poss t
  shows get-label (label-term α n t|-p) = Some (α, n + size p)
  using assms proof(induct t arbitrary: n p)
  case (Fun f ts)
  show ?case proof(cases p)
    case (Cons i p')
    with Fun(2) have i:i < length ts and p':p' ∈ fun-poss (ts!i) by simp+
    with Fun(1) have get-label (label-term α (n+1) (ts!i) |- p') = Some (α, n +
1 + size p') by simp
    then show ?thesis unfolding Cons label-term.simps subt-at.simps using i by
auto
  qed simp
qed simp

lemma linear-label-term:
  assumes linear-term t
  shows linear-term (label-term α n t)
  using assms proof(induct t arbitrary:n)
  case (Fun f ts)
  from Fun(2) have (is-partition (map vars-term ts))
    by simp
```

```
  then have is-partition (map vars-term (map (label-term α (Suc n)) ts))
   by (metis (mono-tags, lifting) length-map map-nth-eq-conv vars-term-labeled-lhs)
  moreover {fix t assume t:t ∈ set ts
    with Fun(2) have linear-term t
      by simp
    with Fun(1) have linear-term (label-term α (Suc n) t)
      using t by blast
  }
  ultimately show ?case unfolding label-term.simps by simp
qed simp

lemma var-term-lab-to-term:
  assumes p ∈ poss t and t|-p = Var x
  shows term-lab-to-term t |-p = Var x
  using assms proof(induct t arbitrary:p)
  case (Fun f ts)
  then obtain i p' where p':i < length ts p = i#p' p' ∈ poss (ts!i)
    by auto
  then show ?case
     unfolding term-lab-to-term.simps p'(2) subt-at.simps using Fun(1,3) p'(2)
by force
qed simp

lemma poss-term-lab-to-term[simp]: poss t = poss (term-lab-to-term t)
  by(induct t) auto

lemma fun-poss-term-lab-to-term[simp]: fun-poss t = fun-poss (term-lab-to-term
t)
  by(induct t) auto

lemma vars-term-list-term-lab-to-term: vars-term-list t = vars-term-list (term-lab-to-term
t)
proof(induct t)
  case (Var x)
  then show ?case
    by (simp add: vars-term-list.simps(1))
next
  case (Fun f ts)
  then show ?case unfolding vars-term-list.simps term-lab-to-term.simps
    by (smt (verit, best) length-map map-eq-conv' nth-map nth-mem)
qed

lemma vars-term-list-term-to-term-lab: vars-term-list (term-to-term-lab t) = vars-term-list
t
proof(induct t)
  case (Var x)
  then show ?case
    by (simp add: vars-term-list.simps(1))
next
```

case (*Fun f ts*)
then show *?case* **unfolding** *vars-term-list.simps term-to-term-lab.simps*
by (*metis* (*mono-tags*, *lifting*) *length-map map-nth-eq-conv nth-mem*)
qed

lemma *linear-term-to-term-lab*:
assumes *linear-term t*
shows *linear-term* (*term-to-term-lab t*)
using *assms* **proof**(*induct t*)
case (*Fun f ts*)
then show *?case* **unfolding** *term-to-term-lab.simps linear-term.simps*
by (*smt* (*verit*, *best*) *imageE length-map list.set-map map-nth-eq-conv set-vars-term-list vars-term-list-term-to-term-lab*)
qed *simp*

lemma *var-poss-list-term-lab-to-term*: *var-poss-list t* = *var-poss-list* (*term-lab-to-term t*)
proof(*induct t*)
case (*Var x*)
then show *?case*
by (*simp add*: *poss-list.simps*(*1*))
next
case (*Fun f ts*)
then have ∗:(*map var-poss-list ts*) = (*map var-poss-list* (*map term-lab-to-term ts*))
by *auto*
then show *?case* **unfolding** *term-lab-to-term.simps var-poss-list.simps length-map* ∗
by *blast*
qed

lemma *label-poss-labeled-lhs*:
assumes *p* ∈ *fun-poss* (*label-term* α *n t*)
shows *p* ∈ *labelposs* (*label-term* α *n t*)
using *assms* **proof**(*induct t arbitrary*:*p n*)
case (*Fun f ts*)
then show *?case* **proof**(*cases p*)
case (*Cons i p*′)
from *Fun*(*2*) **have** *i*:*i* < *length ts*
unfolding *Cons* **by** *simp*
with *Fun*(*2*) **have** *p*′ ∈ *fun-poss* (*label-term* α (*n*+*1*) (*ts*!*i*))
unfolding *Cons* **by** *auto*
with *i* **have** *p*′ ∈ *labelposs* (*label-term* α (*n*+*1*) (*ts*!*i*))
using *Fun*(*1*) **by** *simp*
with *i* **show** *?thesis*
unfolding *Cons label-term.simps labelposs.simps* **by** *simp*
qed *simp*
qed *simp*

lemma *labeled-var*:
assumes *source A = Var x*
shows *labeled-source A = Var x*
using *assms* **proof**(*induct A*)
case (*Prule α As*)
then show *?case* **proof**(*cases As = []*)
case *True*
from *Prule(2)* **have** *lhs α = Var x*
unfolding *source.simps True list.map* **by** *simp*
with *True* **show** *?thesis*
by *simp*
next
case *False*
then obtain *a as* **where** *as:As = a # as*
using *list.exhaust* **by** *blast*
from *Prule(2)* **obtain** *y* **where** *y:lhs α = Var y*
using *is-Var-def* **by** *fastforce*
from *Prule(2)* **have** *source a = Var x*
unfolding *source.simps y as single-var* **by** *simp*
with *Prule(1) as* **have** *labeled-source a = Var x*
by *simp*
then show *?thesis*
unfolding *labeled-source.simps as y single-var* **by** *simp*
qed
qed *simp-all*

lemma *labelposs-subs-fun-poss*: *labelposs t ⊆ fun-poss t*
proof(*induct t*)
case (*Fun fl ts*)
then show *?case* **proof**(*cases snd fl*)
case *None*
then obtain *f* **where** *f:fl = (f, None)*
by (*metis prod.collapse*)
then have *labelposs (Fun fl ts) = (⋃ i<length ts. {i # p |p. p ∈ labelposs (ts ! i)})*
by *simp*
also have *... ⊆ (⋃ i<length ts. {i # p |p. p ∈ fun-poss (ts ! i)})* **using** *Fun*
by (*smt SUP-mono basic-trans-rules(31) lessThan-iff mem-Collect-eq nth-mem subsetI*)
finally show *?thesis*
by *auto*
next
case (*Some l*)
then obtain *f* **where** *f:fl = (f, Some l)*
by (*metis prod.collapse*)
then have *labelposs (Fun fl ts) = {[]} ∪ (⋃ i<length ts. {i # p |p. p ∈ labelposs (ts ! i)})*
by *simp*
also have *... ⊆ {[]} ∪ (⋃ i<length ts. {i # p |p. p ∈ fun-poss (ts ! i)})* **using**

Fun
by (*smt SUP-mono basic-trans-rules*(*31*) *lessThan-iff mem-Collect-eq nth-mem subsetI sup-mono*)
finally show *?thesis*
by *auto*
qed
qed *simp*

lemma *labelposs-subs-poss*[*simp*]: *labelposs t* ⊆ *poss t*
using *labelposs-subs-fun-poss fun-poss-imp-poss* **by** *blast*

lemma *get-label-imp-labelposs*:
assumes *p* ∈ *poss t* **and** *get-label* (*t*|-*p*) ≠ *None*
shows *p* ∈ *labelposs t*
using *assms* **proof**(*induct p arbitrary*:*t*)
case *Nil*
then show *?case* **unfolding** *subt-at.simps*
by (*smt UnCI get-label.elims insert-iff labelposs.elims prod.sel*(*2*) *term.distinct*(*1*) *term.inject*(*2*))
next
case (*Cons i p*)
then obtain *f ts* **where** *t*:*t* = *Fun f ts* **and** *p* ∈ *poss* (*ts* ! *i*) **and** *i*:*i* < *length ts*
using *args-poss* **by** *blast*
with *Cons*(*1*,*3*) **have** *p* ∈ *labelposs* (*ts*!*i*)
by *simp*
with *i* **have** *p*:*i* # *p* ∈ (⋃*i*<*length ts*. {*i* # *p* |*p*. *p* ∈ *labelposs* (*ts* ! *i*)})
by *blast*
then show *?case* **proof**(*cases snd f*)
case *None*
with *p* **show** *?thesis* **unfolding** *t* **using** *labelposs.simps*(*2*)
by (*metis* (*mono-tags*, *lifting*) *prod.collapse*)
next
case (*Some a*)
with *p* **show** *?thesis* **unfolding** *t* **using** *labelposs.simps*(*3*)
by (*smt UN-iff Un-iff mem-Collect-eq prod.collapse*)
qed
qed

lemma *labelposs-obtain-label*:
assumes *p* ∈ *labelposs t*
shows ∃*α m*. *get-label* (*t*|-*p*) = *Some*(*α*, *m*)
using *assms* **proof**(*induct t arbitrary*: *p*)
case (*Fun fl ts*)
then show *?case* **proof**(*cases p*)
case *Nil*
{fix *f* **assume** *f*:*fl* = (*f*, *None*)
from *Fun*(*2*) **have** *False* **unfolding** *Nil f labelposs.simps*(*2*)
by *blast*
}

```
    with Nil show ?thesis
      by (metis eq-snd-iff get-label.simps(2) option.exhaust subt-at.simps(1))
  next
    case (Cons i q)
    with Fun(2) have iq:i # q ∈ labelposs (Fun fl ts)
      by simp
    then have i:i < length ts
      using labelposs-subs-poss by fastforce
    with iq have i # q ∈ {i # p |p. p ∈ labelposs (ts ! i)} proof(cases snd fl)
      case (Some a)
      then obtain f α n where f:fl = (f, Some (α, n))
        by (metis eq-snd-iff)
      from iq show ?thesis unfolding f labelposs.simps
        by blast
    qed (smt UN-iff labelposs.simps(2) list.inject mem-Collect-eq prod.collapse)
    with i Fun(1) Cons show ?thesis
      by simp
  qed
qed simp

lemma possL-obtain-label:
  assumes p ∈ possL A
  shows ∃α m. get-label ((labeled-source A)|-p) = Some(α, m)
  using assms labelposs-obtain-label by blast

lemma labeled-source-apply-subst:
  assumes A ∈ wf-pterm R
  shows labeled-source (A · σ) = (labeled-source A) · (labeled-source ∘ σ)
using assms proof(induct A)
  case (3 α As)
  have id:∀ x ∈ vars-term (labeled-lhs α). (⟨map (labeled-source ∘ (λt. t · σ)) As⟩α)
x = (⟨map labeled-source As⟩α ∘s (labeled-source ∘ σ)) x
  proof−
    have vars:vars-term (labeled-lhs α) = set (var-rule α) using vars-term-labeled-lhs
by simp
    { fix i assume i:i < length (var-rule α)
      with 3 have (⟨map (labeled-source ∘ (λt. t · σ)) As⟩α) ((var-rule α)!i) =
labeled-source ((As!i) · σ)
        by (simp add: mk-subst-distinct)
      also have ... = labeled-source (As!i) · (labeled-source ∘ σ)
        using 3 i by (metis nth-mem)
      also have ... = (⟨map labeled-source As⟩α ∘s (labeled-source ∘ σ)) ((var-rule
α)!i)
        using 3 i unfolding subst-compose-def by (simp add: mk-subst-distinct)
      finally have (⟨map (labeled-source ∘ (λt. t · σ)) As⟩α) ((var-rule α)!i) =
(⟨map labeled-source As⟩α ∘s (labeled-source ∘ σ)) ((var-rule α)!i) .
    } with vars show ?thesis by (metis in-set-idx)
  qed
  have labeled-source ((Prule α As) · σ) = (labeled-lhs α) · ⟨map (labeled-source ∘
```

$(\lambda t.\ t \cdot \sigma))\ As\rangle_\alpha$
unfolding *eval-term.simps*(*2*) **by** *simp*
also have ... = (*labeled-lhs* α) $\cdot$ ($\langle$*map labeled-source As*$\rangle_\alpha$ $\circ_s$ (*labeled-source* $\circ$ σ))
using *id* **by** (*meson term-subst-eq*)
also have ... = (*labeled-source* (*Prule* α *As*)) $\cdot$ (*labeled-source* $\circ$ σ) **by** *simp*
finally show *?case* **.**
qed *simp-all*

lemma *labelposs-apply-subst*:
labelposs ($s \cdot \sigma$) = *labelposs s* $\cup$ {*p@q*| *p q x. p* $\in$ *var-poss s* $\wedge$ *s*|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}
proof(*induct s*)
case (*Fun f ts*)
obtain *f'* *l* **where** *f*:*f* = (*f'*, *l*) **by** *fastforce*
let *?lp1*=$\bigcup$ *i*<*length ts*. {*i* # *p* |*p. p* $\in$ *labelposs* (*ts* ! *i*)}
let *?lp2*=$\bigcup$ *i*<*length ts*. {*i*#(*p@q*)| *p q x. p* $\in$ *var-poss* (*ts*!*i*) $\wedge$ (*ts*!*i*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}
{**fix** *i* **assume** *i*:*i* < *length ts*
with *Fun* **have** {*i* # *p* |*p. p* $\in$ *labelposs* (*ts* ! *i* $\cdot$ σ)} = {*i* # *p*| *p. p* $\in$ *labelposs* (*ts*!*i*) $\cup$ {*p@q*| *p q x. p* $\in$ *var-poss* (*ts*!*i*) $\wedge$ (*ts*!*i*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}}
by *auto*
then have {*i* # *p* |*p. p* $\in$ *labelposs* (*map* ($\lambda s.\ s \cdot \sigma$) *ts* ! *i*)} = {*i* # *p*| *p. p* $\in$ *labelposs* (*ts*!*i*)} $\cup$ {*i*#(*p@q*)| *p q x. p* $\in$ *var-poss* (*ts*!*i*) $\wedge$ (*ts*!*i*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}
unfolding *Un-iff* **using** *i* **by** *auto*
}**note** *IH*=*this*
{**fix** *i* **assume** *i*:*i* < *length ts*
let *?l*={*i*#(*p@q*)| *p q x. p* $\in$ *var-poss* (*ts*!*i*) $\wedge$ (*ts*!*i*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}
let *?r*={*p@q*| *p q x. p* $\in$ {*i* # *p* |*p. p* $\in$ *var-poss* (*ts* ! *i*)} $\wedge$ (*Fun f ts*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)}
have *?l* = *?r* **proof**
show *?l* $\subseteq$ *?r*
by (*smt* (*verit*, *ccfv-SIG*) *Collect-mono-iff Cons-eq-appendI mem-Collect-eq subt-at.simps*(*2*))
show *?r* $\subseteq$ *?l*
by (*smt* (*verit*, *best*) *Collect-mono-iff Cons-eq-appendI mem-Collect-eq subt-at.simps*(*2*))
qed
}
then have *lp2*:{*p@q*| *p q x. p* $\in$ *var-poss* (*Fun f ts*) $\wedge$ (*Fun f ts*)|-*p* = *Var x* $\wedge$ *q* $\in$ *labelposs* (σ *x*)} = *?lp2*
unfolding *var-poss.simps* **by** *auto*
show *?case* **proof**(*cases l*)
case *None*
have *labelposs* (*Fun f ts* $\cdot$ σ) = *?lp1* $\cup$ *?lp2*
unfolding *eval-term.simps f None labelposs.simps length-map* **using** *IH* **by** *auto*

```
    moreover have labelposs (Fun f ts) = ?lp1
      unfolding f None by simp
    ultimately show ?thesis using lp2 by simp
  next
    case (Some a)
    have labelposs (Fun f ts · σ) = {[]} ∪ ?lp1 ∪ ?lp2
      unfolding eval-term.simps f Some labelposs.simps length-map using IH by
auto
    moreover have labelposs (Fun f ts) = {[]} ∪ ?lp1
      unfolding f Some by simp
    ultimately show ?thesis using lp2 by simp
  qed
qed simp

lemma possL-apply-subst:
  assumes A · σ ∈ wf-pterm R
  shows possL (A · σ) = possL A ∪ {p@q| p q x. p ∈ var-poss (labeled-source A)
∧ (labeled-source A)|-p = Var x ∧ q ∈ possL (σ x)}
proof−
  from assms have ∗:labeled-source (A · σ) = labeled-source A · (labeled-source ∘
σ)
    using labeled-source-apply-subst subst-imp-well-def by blast
  then show ?thesis unfolding ∗ labelposs-apply-subst
    by auto
qed

lemma label-term-to-term[simp]: term-lab-to-term (label-term α n t) = t
  by(induct t arbitrary:α n)(simp-all add: map-nth-eq-conv)

lemma fun-poss-label-term: p ∈ fun-poss (label-term β n t) ⟷ p ∈ fun-poss t
proof
  {assume p ∈ fun-poss (label-term β n t)
    then show p ∈ fun-poss t proof(induct t arbitrary:n p)
      case (Fun f ts)
      then show ?case by(cases p) auto
    qed simp
  }
  {assume p ∈ fun-poss t
    then show p ∈ fun-poss (label-term β n t) proof(induct t arbitrary:n p)
      case (Fun f ts)
      then show ?case by(cases p) auto
    qed simp
  }
qed

lemma term-lab-to-term-subst: term-lab-to-term (t · σ) = term-lab-to-term t ·
(term-lab-to-term ∘ σ)
proof(induct t)
  case (Fun f As)
```

then show *?case* **unfolding** *eval-term.simps*(*2*) *term-lab-to-term.simps*
by *fastforce*
qed *simp*

lemma *labeled-source-to-term*[*simp*]: *term-lab-to-term* (*labeled-source A*) = *source A*
proof(*induct A*)
case (*Prule α As*)
have *term-lab-to-term* ∘ ⟨*map labeled-source As*⟩$_\alpha$ = ⟨*map* (*term-lab-to-term* ∘ *labeled-source*) *As*⟩$_\alpha$
by *simp*
also have ... = ⟨*map source As*⟩$_\alpha$ **using** *Prule*
by (*metis* (*mono-tags*, *lifting*) *comp-apply map-eq-conv*)
finally show *?case* **unfolding** *labeled-source.simps source.simps*
by (*simp add*: *term-lab-to-term-subst*)
qed *simp-all*

lemma *possL-subset-poss-source*: *possL A* ⊆ *poss* (*source A*)
using *poss-term-lab-to-term labeled-source-to-term labelposs-subs-poss*
by *metis*

lemma *labeled-source-pos*:
assumes *p* ∈ *poss s* **and** *term-lab-to-term t* = *s*
shows *term-lab-to-term* (*t*|-*p*) = *s*|-*p*
using *assms* **proof**(*induct p arbitrary*:*s t*)
case (*Cons i p*)
from *Cons*(*2*) **obtain** *f ss* **where** *s*:*s* = *Fun f ss*
using *args-poss* **by** *blast*
with *Cons*(*2*) **have** *p*:*p* ∈ *poss* (*ss*!*i*)
by *force*
from *Cons*(*3*) *s* **obtain** *label ts* **where** *t*:*t* = *Fun* (*f*, *label*) *ts*
by (*metis args-poss local.Cons*(*2*) *poss-term-lab-to-term prod.collapse term.inject*(*2*) *term-lab-to-term.simps*(*2*))
with *Cons*(*2*,*3*) *s* **have** *term-lab-to-term* (*ts*!*i*) = *ss*!*i*
by *auto*
with *Cons*(*1*) *p* **show** *?case* **unfolding** *s t*
by *simp*
qed *simp*

lemma *get-label-at-fun-poss-subst*:
assumes *p* ∈ *fun-poss t*
shows *get-label* ((*t* · *σ*)|-*p*) = *get-label* (*t*|-*p*)
using *assms fun-poss-fun-conv fun-poss-imp-poss* **by** *fastforce*

lemma *labeled-source-simple-pterm*:*possL* (*to-pterm t*) = {}
by(*induct t*) *simp-all*

lemma *label-term-increase*:
assumes *s* = (*label-term α n t*) · *σ* **and** *p* ∈ *fun-poss t*

```
  shows get-label (s|-p) = Some (α, n + length p)
  using assms proof(induct p arbitrary: s t n)
  case Nil
  then obtain f ts where t = Fun f ts
   by (metis fun-poss-list.simps(1) in-set-simps(3) is-FunE is-Var-def set-fun-poss-list)
  with Nil(1) show ?case
    by simp
next
  case (Cons i p)
  then obtain f ts where f:t = Fun f ts and i:i<length ts
    by (meson args-poss fun-poss-imp-poss)
  with Cons(3) have p:p ∈ fun-poss (ts!i)
    by auto
  let ?s' = (label-term α (n+1) (ts!i)) · σ
  from Cons(1) p have get-label (?s'|- p) = Some (α, n + 1 + length p)
    by blast
  with i show ?case unfolding Cons(2) f
    by simp
qed
```

The number attached to a labeled function symbol cannot exceed the depth of that function symbol.

```
lemma label-term-max-value:
  assumes p ∈ poss (labeled-source A) and get-label ((labeled-source A)|-p) = Some
(α, n)
    and A ∈ wf-pterm R
  shows n ≤ length p
  using assms proof(induct A arbitrary: p)
  case (Pfun f As)
  then show ?case proof(cases p)
    case (Cons i q)
    with Pfun(2) have i:i < length As by simp
    with Pfun(3) have lab:get-label (labeled-source (As!i) |- q) = Some (α, n)
      unfolding Cons by simp
    with Pfun(2) i have q ∈ poss (labeled-source (As!i))
      unfolding Cons by auto
    with Pfun(1,4) Cons i lab show ?thesis
      using nth-mem fun-well-arg by fastforce
  qed simp
next
  case (Prule β As)
  from Prule(2) consider p ∈ fun-poss (labeled-lhs β) | (∃ p1 p2 x. p = p1@p2
                                  ∧ p1 ∈ poss (labeled-lhs β) ∧ (labeled-lhs β)|-p1 =
Var x
                                  ∧ p2 ∈ poss ((⟨map labeled-source As⟩β) x)
                                      ∧ (labeled-source (Prule β As))|-p = ((⟨map
labeled-source As⟩β) x)|-p2)
   unfolding labeled-source.simps by (meson poss-is-Fun-fun-poss poss-subst-choice)
  then show ?case proof(cases)
```

```
    case 1
    then have p ∈ fun-poss (lhs β)
      by (simp add: fun-poss-label-term)
    then have get-label ((labeled-source (Prule β As))|-p) = Some (β, length p)
      unfolding labeled-source.simps by (simp add: label-term-increase)
    with Prule(3) show ?thesis by auto
  next
    case 2
    then obtain p1 p2 x where p1p2:p = p1 @ p2 and x:p1 ∈ poss (labeled-lhs
β) ∧ labeled-lhs β |- p1 = Var x
       and p2:p2 ∈ poss ((⟨map labeled-source As⟩β) x)
       and lab:labeled-source (Prule β As) |- p = (⟨map labeled-source As⟩β) x |- p2
      by blast
    from Prule(4) have l:length As = length (var-rule β)
      using wf-pterm.simps by fastforce
    from x have x ∈ vars-term (lhs β)
    by (metis subt-at-imp-supteq subteq-Var-imp-in-vars-term vars-term-labeled-lhs)
    with x obtain i where i:i < length (var-rule β) ∧ (var-rule β)!i = x
      by (metis in-set-conv-nth set-vars-term-list vars-term-list-vars-distinct)
    with l have ∗:(⟨map labeled-source As⟩β) x = labeled-source (As!i)
      by (metis (no-types, lifting) length-map lhs-subst-var-i nth-map)
    with Prule(3) lab have get-label ((labeled-source (As!i))|-p2) = Some (α, n)
      by simp
    with Prule(1,4) p2 ∗ i l have n ≤ length p2
      by (metis fun-well-arg nth-mem)
    with ∗ p1p2 lab i l show ?thesis by force
  qed
qed simp
```

The labels decrease when moving up towards the root from a labeled function symbol.

```
lemma label-decrease:
  assumes p@q ∈ poss (labeled-source A)
    and get-label ((labeled-source A)|-(p@q)) = Some (α, length q + n)
    and A ∈ wf-pterm R
  shows get-label ((labeled-source A)|-p) = Some (α, n)
  using assms proof(induct A arbitrary: p q)
  case (Pfun f As)
  then show ?case proof(cases p)
    case Nil
    from Pfun(2,3) obtain i q' where iq':q = i#q' and i:i < length As and q':q'
∈ poss (labeled-source (As!i))
      unfolding Nil by auto
    with Pfun(2,3) have get-label (labeled-source (As!i) |- (q')) = Some (α, length
q + n)
      unfolding Nil by auto
    with iq' q' have False
      using label-term-max-value Pfun(4) i fun-well-arg by (metis le-imp-less-Suc
length-nth-simps(2) not-add-less1 nth-mem)
```

```
    then show ?thesis by simp
  next
    case (Cons i p')
    with Pfun(2) have ip':p = i#p' and i:i < length As
      by auto
    with Pfun(2) have p':p'@q ∈ poss (labeled-source (As!i))
      by simp
    from Pfun(3) i ip' have get-label (labeled-source (As!i) |- (p'@q)) = Some (α,
length q + n)
      by simp
    with Pfun(1,4) p' i have get-label ((labeled-source (As!i))|-p') = Some (α, n)
      by (metis fun-well-arg nth-mem)
    then show ?thesis
      using i ip' by fastforce
  qed
next
  case (Prule β As)
  from Prule(2) consider p@q ∈ fun-poss (labeled-lhs β) | (∃ p1 p2 x. p@q =
p1@p2
                                  ∧ p1 ∈ poss (labeled-lhs β) ∧ (labeled-lhs β)|-p1 =
Var x
                                  ∧ p2 ∈ poss ((⟨map labeled-source As⟩β) x)
                                  ∧ (labeled-source (Prule β As))|-(p@q) = ((⟨map
labeled-source As⟩β) x)|-p2)
    unfolding labeled-source.simps by (meson poss-is-Fun-fun-poss poss-subst-choice)
  then show ?case proof(cases)
    case 1
     then have lab:get-label ((labeled-source (Prule β As))|-(p@q)) = Some (β,
length p + length q)
      by (simp add: fun-poss-label-term label-term-increase)
    from 1 have p ∈ fun-poss (labeled-lhs β) proof(cases q)
      case (Cons a list)
      then show ?thesis
        using 1 fun-poss-append-poss fun-poss-imp-poss by blast
    qed simp
    with Prule(3) lab show ?thesis
      by (simp add: fun-poss-label-term label-term-increase)
  next
    case 2
   then obtain p1 p2 x where p1p2:p@q = p1 @ p2 and x:p1 ∈ poss (labeled-lhs
β) ∧ labeled-lhs β |- p1 = Var x
       and p2:p2 ∈ poss ((⟨map labeled-source As⟩β) x)
       and lab:labeled-source (Prule β As) |-(p@q) = (⟨map labeled-source As⟩β) x
|- p2
      by blast
    from Prule(4) have l:length As = length (var-rule β)
      using wf-pterm.simps by fastforce
    from x have x ∈ vars-term (lhs β)
    by (metis subt-at-imp-supteq subteq-Var-imp-in-vars-term vars-term-labeled-lhs)
```

then obtain i **where** i:$i < length\ (var\text{-}rule\ \beta) \wedge (var\text{-}rule\ \beta)!i = x$
by (*metis in-set-idx set-vars-term-list vars-term-list-vars-distinct*)
with l **have** $*$:$(\langle map\ labeled\text{-}source\ As\rangle_\beta)\ x = labeled\text{-}source\ (As!i)$
by (*metis* (*no-types*, *lifting*) *length-map lhs-subst-var-i nth-map*)
with *Prule*(*3*) *lab* **have** *as-i*:*get-label* ((*labeled-source* (*As*!*i*))|-*p2*) = *Some* (α, *length q* + *n*)
by *simp*
have *p1-above-p*:*p1* $\leq_p$ *p* **proof**(*rule ccontr*)
assume ¬ *p1* $\leq_p$ *p*
with *p1p2* **have** *length p* < *length p1*
by (*metis less-eq-pos-simps*(*1*) *pos-cases pos-less-eq-append-not-parallel prefix-smaller*)
with *p1p2* **have** *le*:*length p2* < *length q*
using *length-append* **by** (*metis add.commute less-add-eq-less*)
with *as-i Prule*(*4*) $*$ *i l p2* **have** *length q* + *n* ≤ *length p2*
by (*metis fun-well-arg label-term-max-value nth-mem*)
with *le* **show** *False* **by** *linarith*
qed
let *?p'*=*the* (*remove-prefix p1 p*)
from *p1-above-p p1p2* **have** *p2'*:*p2* = *?p'* @ *q*
by (*metis append-assoc pos-diff-def prefix-pos-diff same-append-eq*)
then have *lab'*:*labeled-source* (*Prule* β *As*) |-(*p1*@*?p'*) = ($\langle$*map labeled-source As*$\rangle_\beta$) *x* |-*?p'*
using *x p1p2 Prule*(*2*) **unfolding** *labeled-source.simps*
by (*metis* (*mono-tags*, *lifting*) *labeled-source.simps*(*3*) *poss-append-poss eval-term.simps*(*1*) *subt-at-subst subterm-poss-conv*)
from *p2' Prule*(*1*,*4*) *p2* $*$ *i l as-i* **have** *get-label* ((*labeled-source* (*As*!*i*))|-*?p'*) = *Some* (α, *n*)
by (*metis fun-well-arg nth-mem*)
with *lab'* $*$ **show** *?thesis*
by (*metis p1-above-p pos-diff-def prefix-pos-diff*)
qed
qed *simp*

If a function symbol is labeled with (α, *n*), then the function symbol *n* positions above it is labeled with (α, *0*).

lemma *obtain-label-root*:
assumes *p* ∈ *poss* (*labeled-source A*)
and *get-label* ((*labeled-source A*)|-*p*) = *Some* (α, *n*)
and *A* ∈ *wf-pterm R*
shows *get-label* ((*labeled-source A*)|-(*take* (*length p* − *n*) *p*)) = *Some* (α, *0*) ∧ *take* (*length p* − *n*) *p* ∈ *poss* (*labeled-source A*)
proof−
from *assms* **have** *n*:*n* ≤ *length p*
using *label-term-max-value* **by** *blast*
with *assms* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *add.right-neutral append-take-drop-id diff-diff-cancel label-decrease length-drop poss-append-poss*)
qed

lemma *label-ctxt-apply-term*:
assumes *get-label* (*labeled-source A* |- *p*) = *l q* ∈ *poss s*
shows *get-label* (*labeled-source* ((*ctxt-of-pos-term q* (*to-pterm s*)) ⟨*A*⟩) |- (*q*@*p*))
= *l*
using *assms*(*2*) **proof**(*induct s arbitrary*:*q*)
case (*Var x*)
then have *q*:*q* = [] **by** *simp*
from *assms*(*1*) **show** *?case* **unfolding** *q* **by** *simp*
next
case (*Fun f ts*)
then show *?case* **proof**(*cases q*)
case *Nil*
from *assms*(*1*) **show** *?thesis* **unfolding** *Nil* **by** *simp*
next
case (*Cons i q*′)
with *Fun*(*2*) **have** *i*:*i* < *length ts* **and** *q*′:*q*′ ∈ *poss* (*ts*!*i*) **by** *auto*
with *Fun*(*1*) **have** *get-label* (*labeled-source* (*ctxt-of-pos-term q*′ (*to-pterm* (*ts*!*i*)))⟨*A*⟩
|- (*q*′ @ *p*)) = *l* **by** *simp*
then show *?thesis*
unfolding *to-pterm.simps Cons ctxt-of-pos-term.simps labeled-source.simps*
append-Cons intp-actxt.simps subt-at.simps
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc append-take-drop-id i length-append*
length-map less-imp-le-nat list.size(*4*) *nth-append-take nth-map*)
qed
qed

lemma *single-redex-at-p-label*:
assumes *p* ∈ *poss s* **and** *is-Fun* (*lhs* α)
shows *get-label* (*labeled-source* (*ll-single-redex s p* α) |-*p*) = *Some*(α, *0*)
proof−
from *assms*(*2*) **obtain** *f ts* **where** *f*:*lhs* α = *Fun f ts*
by *blast*
have *get-label* (*labeled-source* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*)))
(*var-poss-list* (*lhs* α))))) = *Some* (α, *0*)
unfolding *f labeled-source.simps label-term.simps eval-term.simps get-label.simps*
by *simp*
then show *?thesis*
unfolding *ll-single-redex-def* **using** *label-ctxt-apply-term*[**where** *p*=[]] *assms*(*1*)
by (*smt* (*verit*) *self-append-conv subt-at.simps*(*1*))
qed

Whenever a function symbol at position p has label $(\alpha, 0)$ or no label in *labeled-source* A, then we know that there exists a position q in A such that A |- $q = \alpha$ *As* for appropriate *As*. Moreover, taking the source of the context at position q must be the same as first computing the source of A and then taking the context at p.

context *left-lin*
begin

lemma *poss-labeled-source*:
assumes $p \in poss\ (labeled\text{-}source\ A)$
and $get\text{-}label\ ((labeled\text{-}source\ A)|\text{-}p) = Some\ (\alpha,\ 0)$
and $A \in wf\text{-}pterm\ R$
shows $\exists\, q \in poss\ A.\ ctxt\text{-}of\text{-}pos\text{-}term\ p\ (source\ A) = source\text{-}ctxt\ (ctxt\text{-}of\text{-}pos\text{-}term\ q\ A) \wedge$
$A|\text{-}q = Prule\ \alpha\ (map\ (\lambda i.\ A|\text{-}(q@[i]))\ [0..<length\ (var\text{-}rule\ \alpha)])$
using *assms* **proof**(*induct A arbitrary*:*p*)
case (*Var x*)
then have $p = []$ **by** *simp*
with *Var*(*2*) **have** *False* **unfolding** *labeled-source.simps* **by** *simp*
then show *?case* **by** *blast*
next
case (*Pfun f As*)
then show *?case* **proof**(*cases p*)
case (*Cons i p'*)
with *Pfun*(*2*) **have** $ip'{:}p = i\#p'$ **and** $i{:}i < length\ As$
by *auto*
with *Pfun*(*2*) **have** $p'{:}p' \in poss\ (labeled\text{-}source\ (As!i))$
by *simp*
from *Pfun*(*3*) *i ip'* **have** $get\text{-}label\ (labeled\text{-}source\ (As!i)\ |\text{-}\ p') = Some\ (\alpha,\ 0)$
by *simp*
with *Pfun*(*1*,*4*) p' *i* **obtain** *q* **where** $q{:}q \in poss\ (As!i)$ **and** $ctxt\text{-}of\text{-}pos\text{-}term\ p'\ (source\ (As!i)) = source\text{-}ctxt\ (ctxt\text{-}of\text{-}pos\text{-}term\ q\ (As!i))$
and $prule{:}(As!i)|\text{-}q = Prule\ \alpha\ (map\ (\lambda j.\ (As!i)|\text{-}(q@[j]))\ [0..<length\ (var\text{-}rule\ \alpha)])$
using *nth-mem* **by** *blast*
then have $ctxt\text{-}of\text{-}pos\text{-}term\ p\ (source\ (Pfun\ f\ As)) = source\text{-}ctxt\ (ctxt\text{-}of\text{-}pos\text{-}term\ (i\#q)\ (Pfun\ f\ As))$
unfolding ip' **using** *i* **by**(*simp add*: *take-map drop-map*)
then show *?thesis*
using *q*(*1*) *i prule* **by** *fastforce*
qed *simp*
next
case (*Prule β As*)
have $l{:}length\ As = length\ (var\text{-}rule\ \beta)$
using *Prule*(*4*) **using** *wf-pterm.simps* **by** *fastforce*
from *Prule*(*2*) **consider** $p \in fun\text{-}poss\ (labeled\text{-}lhs\ \beta) \mid (\exists\, p1\ p2\ x.\ p = p1@p2 \wedge p1 \in poss\ (labeled\text{-}lhs\ \beta) \wedge (labeled\text{-}lhs\ \beta)|\text{-}p1 = Var\ x \wedge p2 \in poss\ ((\langle map\ labeled\text{-}source\ As\rangle_\beta)\ x) \wedge (labeled\text{-}source\ (Prule\ \beta\ As))|\text{-}p = ((\langle map\ labeled\text{-}source\ As\rangle_\beta)\ x)|\text{-}p2)$
unfolding *labeled-source.simps* **by** (*meson poss-is-Fun-fun-poss poss-subst-choice*)
then show *?case* **proof**(*cases*)
case *1*
then have $p \in fun\text{-}poss\ (lhs\ \beta)$
by (*simp add*: *fun-poss-label-term*)
then have $get\text{-}label\ ((labeled\text{-}source\ (Prule\ \beta\ As))|\text{-}p) = Some\ (\beta,\ length\ p)$

unfolding *labeled-source.simps* **by** (*simp add*: *label-term-increase*)
with *Prule*(*3*) **have** *p*:*p* = [] **and** α:α = β **by** *simp*+
have *As* = (*map* (λ*i*. *Prule* β *As* |- ([*i*])) [*0*..<*length As*])
by (*simp add*: *map-nth*)
then have *As* = (*map* (λ*i*. *Prule* β *As* |- ([] @ [*i*])) [*0*..<*length* (*var-rule* α)])
unfolding α **using** *l* **by** *force*
then show *?thesis* **unfolding** *p* α **by** *auto*
next
case *2*
then obtain *p1 p2 x* **where** *p1p2*:*p* = *p1* @ *p2* **and** *x*:*p1* ∈ *poss* (*labeled-lhs* β) ∧ *labeled-lhs* β |- *p1* = *Var x*
and *p2*:*p2* ∈ *poss* ((⟨*map labeled-source As*⟩$_\beta$) *x*)
and *lab*:*labeled-source* (*Prule* β *As*) |- *p* = ((⟨*map labeled-source As*⟩$_\beta$) *x* |- *p2*
by *blast*
from *Prule*(*4*) **have** *l*:*length As* = *length* (*var-rule* β)
using *wf-pterm.simps* **by** *fastforce*
from *Prule*(*4*) **have** *to-rule* β ∈ *R*
using *wf-pterm.cases* **by** *force*
with *left-lin* **have** *lin*:*linear-term* (*lhs* β)
using *left-linear-trs-def* **by** *fastforce*
from *x* **have** *p1*:*p1* ∈ *poss* (*lhs* β) **by** *simp*
then have *p1'*:*p1* ∈ *poss* ((*lhs* β) · ⟨*map source As*⟩$_\beta$) **by** *simp*
from *p1 x* **have** *x'*:*lhs* β |- *p1* = *Var x*
by (*metis label-term-to-term labeled-source-pos term-lab-to-term.simps*(*1*))
from *p1 x'* **obtain** *i* **where** *i*:*i* < *length* (*vars-term-list* (*lhs* β)) *var-poss-list* (*lhs* β) ! *i* = *p1 vars-term-list* (*lhs* β) ! *i* = *x*
by (*metis in-set-idx length-var-poss-list term.inject*(*1*) *var-poss-iff var-poss-list-sound vars-term-list-var-poss-list*)
with *lin* **have** *i'*:*i* < *length* (*var-rule* β) ∧ (*var-rule* β)!*i* = *x*
by (*metis linear-term-var-vars-term-list*)
with *l* **have** ∗:(⟨*map labeled-source As*⟩$_\beta$) *x* = *labeled-source* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *length-map lhs-subst-var-i nth-map*)
with *Prule*(*3*) *lab* **have** *get-label* ((*labeled-source* (*As*!*i*))|-*p2*) = *Some* (α, *0*)
by *simp*
with *Prule*(*1*,*4*) *p2* ∗ *i' l* **obtain** *q* **where** *q*:*q*∈*poss* (*As*!*i*) *ctxt-of-pos-term p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term q* (*As*!*i*))
(*As*!*i*) |- *q* = *Prule* α (*map* (λ*j*. (*As*!*i*) |- (*q* @ [*j*])) [*0*..<*length* (*var-rule* α)])
by (*smt* (*verit*, *ccfv-SIG*) *fun-well-arg map-eq-conv nth-mem*)
have *p1''*:(*var-poss-list* (*lhs* β) ! *length* (*take i As*)) = *p1*
using *i l* **by** (*metis id-take-nth-drop length-take length-var-poss-list lin linear-term-var-vars-term-list nth-append-length*)
have *x-sub*:*Var x* · ⟨*map source As*⟩$_\beta$ = *source* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *i' l length-map lhs-subst-var-i nth-map eval-term.simps*(*1*))
have *ctxt-of-pos-term p* (*source* (*Prule* β *As*)) = *source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Prule* β *As*)) **proof**−
{**fix** *y* **assume** *y* ∈ *vars-term* (*lhs* β) *y* ≠ *vars-term-list* (*lhs* β) ! *i*
then obtain *j* **where** *j*:*j* < *length* (*var-rule* β) *y* = (*var-rule* β) ! *j j* ≠ *i*
by (*metis in-set-conv-nth lin linear-term-var-vars-term-list set-vars-term-list*)
have *x*:(*vars-term-list* (*lhs* β) ! *length* (*take i As*)) = *x*

by (*metis i′ id-take-nth-drop l length-take lin linear-term-var-vars-term-list nth-append-length*)
from *j* **have** (⟨*map source* (*take i As* @ *Var x* # *drop* (*Suc i*) *As*)⟩$_\beta$) *y* = *source* (*As*!*j*)
using *apply-lhs-subst-var-rule l*
by (*smt* (*verit*, *best*) *Cons-nth-drop-Suc append-Cons-nth-not-middle append-take-drop-id i′ length-append length-map length-nth-simps*(2) *lin linear-term-var-vars-term-list nth-map x*)
then have (⟨*map source* (*take i As* @ *Var* (*vars-term-list* (*lhs* β) ! *length* (*take i As*)) # *drop* (*Suc i*) *As*)⟩$_\beta$) *y* = (⟨*map source As*⟩$_\beta$) *y*
unfolding *x* **by** (*metis* (*no-types*, *lifting*) *j*(1,2) *l length-map lhs-subst-var-i nth-map*)
}
then have ∗:*ctxt-of-pos-term p1* (*lhs* β) ·$_c$ ⟨*map source As*⟩$_\beta$ =
ctxt-of-pos-term p1 (*lhs* β · ⟨*map source* (*take i As* @ *Var* (*vars-term-list* (*lhs* β) ! *length* (*take i As*)) # *drop* (*Suc i*) *As*)⟩$_\beta$)
using *i*
unfolding *ctxt-of-pos-term-subst*[*OF p1*, *symmetric*]
apply (*intro ctxt-of-pos-term-hole-subst*[*OF lin*, *of i*])
subgoal by (*metis length-var-poss-list*)
by *auto*
then show *?thesis*
unfolding *source.simps p1p2 ctxt-of-pos-term-append*[*OF p1* ‵] *ctxt-of-pos-term-subst*[*OF p1*] *subt-at-subst*[*OF p1*] *x′ ctxt-of-pos-term.simps source-ctxt.simps Let-def x-sub q*(2) ∗ *p1′′*
by *simp*
qed
moreover from *q*(3) **have** *Prule* β *As* |- (*i*#*q*) = *Prule* α (*map* (λ*j*. *Prule* β *As* |- ((*i*#*q*) @ [*j*])) [*0*..<*length* (*var-rule* α)])
by *simp*
ultimately show *?thesis*
using *i′ q*(1) *l* **by** (*metis poss-Cons-poss term.sel*(4))
qed
qed

lemma *poss-labeled-source-None*:
assumes *p* ∈ *poss* (*labeled-source A*)
and *get-label* ((*labeled-source A*)|-*p*) = *None*
and *A* ∈ *wf-pterm R*
shows ∃ *q* ∈ *poss A*. *ctxt-of-pos-term p* (*source A*) = *source-ctxt* (*ctxt-of-pos-term q A*)
using *assms* **proof**(*induct A arbitrary*:*p*)
case (*Pfun f As*)
then show *?case* **proof**(*cases p*)
case (*Cons i p′*)
with *Pfun*(2) **have** *ip′*:*p* = *i*#*p′* **and** *i*:*i* < *length As*
by *auto*
with *Pfun*(2) **have** *p′*:*p′* ∈ *poss* (*labeled-source* (*As*!*i*))
by *simp*

from *Pfun*(*3*) **have** *get-label* (*labeled-source* (*As* ! *i*) |- *p'*) = *None*
unfolding *ip' labeled-source.simps* **using** *i* **by** *simp*
with *Pfun*(*1*,*4*) *p' i* **obtain** *q* **where** *q*:*q* ∈ *poss* (*As*!*i*) **and** *ctxt-of-pos-term p'* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term q* (*As*!*i*))
using *nth-mem* **by** *blast*
then have *ctxt-of-pos-term p* (*source* (*Pfun f As*)) = *source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Pfun f As*))
unfolding *ip'* **using** *i* **by**(*simp add*: *take-map drop-map*)
then show *?thesis*
using *q*(*1*) *i* **by** *fastforce*
qed *simp*
next
case (*Prule β As*)
have *l*:*length As* = *length* (*var-rule β*)
using *Prule*(*4*) **using** *wf-pterm.simps* **by** *fastforce*
from *Prule*(*2*) **consider** *p* ∈ *fun-poss* (*labeled-lhs β*) | (∃ *p1 p2 x*. *p* = *p1*@*p2*
∧ *p1* ∈ *poss* (*labeled-lhs β*) ∧ (*labeled-lhs β*)|-*p1* = *Var x*
∧ *p2* ∈ *poss* ((⟨*map labeled-source As*⟩$_\beta$) *x*)
∧ (*labeled-source* (*Prule β As*))|-*p* = ((⟨*map labeled-source As*⟩$_\beta$) *x*)|-*p2*)
unfolding *labeled-source.simps* **by** (*meson poss-is-Fun-fun-poss poss-subst-choice*)
then show *?case* **proof**(*cases*)
case *1*
then have *p* ∈ *fun-poss* (*lhs β*)
by (*simp add*: *fun-poss-label-term*)
then have *get-label* ((*labeled-source* (*Prule β As*))|-*p*) = *Some* (*β*, *length p*)
unfolding *labeled-source.simps* **by** (*simp add*: *label-term-increase*)
then show *?thesis*
using *Prule*(*3*) **by** *simp*
next
case *2*
then obtain *p1 p2 x* **where** *p1p2*:*p* = *p1* @ *p2* **and** *x*:*p1* ∈ *poss* (*labeled-lhs β*) ∧ *labeled-lhs β* |- *p1* = *Var x*
and *p2*:*p2* ∈ *poss* ((⟨*map labeled-source As*⟩$_\beta$) *x*)
and *lab*:*labeled-source* (*Prule β As*) |- *p* = (⟨*map labeled-source As*⟩$_\beta$) *x* |- *p2*
by *blast*
from *Prule*(*4*) **have** *l*:*length As* = *length* (*var-rule β*)
using *wf-pterm.simps* **by** *fastforce*
from *Prule*(*4*) **have** *to-rule β* ∈ *R*
using *wf-pterm.cases* **by** *force*
with *left-lin* **have** *lin*:*linear-term* (*lhs β*)
using *left-linear-trs-def* **by** *fastforce*
from *x* **have** *p1*:*p1* ∈ *poss* (*lhs β*) **by** *simp*
then have *p1'*:*p1* ∈ *poss* ((*lhs β*) · ⟨*map source As*⟩$_\beta$) **by** *simp*
from *p1 x* **have** *x'*:*lhs β* |- *p1* = *Var x*
by (*metis label-term-to-term labeled-source-pos term-lab-to-term.simps*(*1*))
from *p1 x'* **obtain** *i* **where** *i*:*i* < *length* (*vars-term-list* (*lhs β*)) *var-poss-list* (*lhs β*) ! *i* = *p1 vars-term-list* (*lhs β*) ! *i* = *x*

by (*metis in-set-idx length-var-poss-list term.inject*(*1*) *var-poss-iff var-poss-list-sound vars-term-list-var-poss-list*)
with *lin* **have** *i'*:*i* < *length* (*var-rule* β) ∧ (*var-rule* β)!*i* = *x*
by (*metis linear-term-var-vars-term-list*)
with *l* **have** ∗:(⟨*map labeled-source As*⟩$_\beta$) *x* = *labeled-source* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *length-map lhs-subst-var-i nth-map*)
with *Prule*(*3*) *lab* **have** *get-label* ((*labeled-source* (*As*!*i*))|*-p2*) = *None*
by *simp*
with *Prule*(*1*,*4*) *p2* ∗ *i' l* **obtain** *q* **where** *q*:*q*∈*poss* (*As*!*i*) *ctxt-of-pos-term p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term q* (*As*!*i*))
by (*smt* (*verit*, *ccfv-SIG*) *fun-well-arg map-eq-conv nth-mem*)
have *p1''*:(*var-poss-list* (*lhs* β) ! *length* (*take i As*)) = *p1*
using *i l* **by** (*metis id-take-nth-drop length-take length-var-poss-list lin linear-term-var-vars-term-list nth-append-length*)
have *x-sub*:*Var x* · ⟨*map source As*⟩$_\beta$ = *source* (*As*!*i*)
by (*metis* (*no-types*, *lifting*) *i' l length-map lhs-subst-var-i nth-map eval-term.simps*(*1*))
have *ctxt-of-pos-term p* (*source* (*Prule* β *As*)) = *source-ctxt* (*ctxt-of-pos-term* (*i*#*q*) (*Prule* β *As*)) **proof**−
{fix *y* **assume** *y* ∈ *vars-term* (*lhs* β) *y* ≠ *vars-term-list* (*lhs* β) ! *i*
then obtain *j* **where** *j*:*j* < *length* (*var-rule* β) *y* = (*var-rule* β) ! *j j* ≠ *i*
by (*metis in-set-conv-nth lin linear-term-var-vars-term-list set-vars-term-list*)
have *x*:(*vars-term-list* (*lhs* β) ! *length* (*take i As*)) = *x*
by (*metis i' id-take-nth-drop l length-take lin linear-term-var-vars-term-list nth-append-length*)
from *j* **have** (⟨*map source* (*take i As* @ *Var x* # *drop* (*Suc i*) *As*)⟩$_\beta$) *y* = *source* (*As*!*j*)
using *apply-lhs-subst-var-rule l*
by (*smt* (*verit*, *best*) *Cons-nth-drop-Suc append-Cons-nth-not-middle append-take-drop-id i' length-append length-map length-nth-simps*(*2*) *lin linear-term-var-vars-term-list nth-map x*)
then have (⟨*map source* (*take i As* @ *Var* (*vars-term-list* (*lhs* β) ! *length* (*take i As*)) # *drop* (*Suc i*) *As*)⟩$_\beta$) *y* = (⟨*map source As*⟩$_\beta$) *y*
unfolding *x* **by** (*metis* (*no-types*, *lifting*) *j*(*1*,*2*) *l length-map lhs-subst-var-i nth-map*)
}
then have ∗:*ctxt-of-pos-term p1* (*lhs* β) ·$_c$ ⟨*map source As*⟩$_\beta$ =
ctxt-of-pos-term p1 (*lhs* β · ⟨*map source* (*take i As* @ *Var* (*vars-term-list* (*lhs* β) ! *length* (*take i As*)) # *drop* (*Suc i*) *As*)⟩$_\beta$)
using *i*
unfolding *ctxt-of-pos-term-subst*[*OF p1*, *symmetric*]
apply (*intro ctxt-of-pos-term-hole-subst*[*OF lin*, *of i*])
subgoal by (*metis length-var-poss-list*)
by *auto*
then show *?thesis*
unfolding *source.simps p1p2 ctxt-of-pos-term-append*[*OF p1* ⌝] *ctxt-of-pos-term-subst*[*OF p1*] *subt-at-subst*[*OF p1*] *x' ctxt-of-pos-term.simps source-ctxt.simps Let-def x-sub q*(*2*) ∗ *p1''*
by *simp*
qed

```
    then show ?thesis
      using i' q(1) l by (metis poss-Cons-poss term.sel(4))
  qed
qed simp
end
```

If we know that some part of a term does not contain labels (i.e., is coming from a simple proof term t) then we know that the label originates below some variable position of t.

```
lemma labeled-source-to-pterm-subst:
  assumes p-pos:p ∈ possL (to-pterm t · σ) and well:∀ x ∈ vars-term t. σ x ∈
wf-pterm R
  shows ∃ p1 p2 x γ. p1 ∈ poss t ∧ t|-p1 = Var x ∧ p1@p2 ≤p p
      ∧ p2 ∈ possL (σ x) ∧ get-label ((labeled-source (σ x))|-p2) = Some (γ, 0)
proof−
  {assume p:p ∈ fun-poss (labeled-source (to-pterm t))
    then have get-label ((labeled-source (to-pterm t))|-p) = None
        using labeled-source-simple-pterm by (metis empty-iff fun-poss-imp-poss
get-label-imp-labelposs)
     moreover have get-label ((labeled-source ((to-pterm t) · σ))|-p) = get-label
((labeled-source (to-pterm t))|-p)
    by (metis get-label-at-fun-poss-subst labeled-source-apply-subst p to-pterm-wf-pterm)
    ultimately have False using p-pos possL-obtain-label by fastforce
  }
 with p-pos obtain p1 r x where p:p = p1@r and p1:p1 ∈ poss t and t:(labeled-source
(to-pterm t))|-p1 = Var x
   by (smt (z3) labeled-source-apply-subst labeled-source-to-term possL-subset-poss-source
poss-subst-apply-term poss-term-lab-to-term source-to-pterm subset-eq to-pterm-wf-pterm)
  then have x:t|-p1 = Var x
   by (metis labeled-source-pos labeled-source-to-term source-to-pterm term-lab-to-term.simps(1))
  from p-pos have r-pos:r ∈ poss (labeled-source (σ x))
    unfolding p using p1 t labeled-source-apply-subst
   by (smt (z3) comp-apply labeled-source-to-term labelposs-subs-poss less-eq-pos-simps(1)
p poss-append-poss poss-term-lab-to-term source-to-pterm subset-eq eval-term.simps(1)
subt-at-subst to-pterm-wf-pterm)
  from p-pos obtain γ n where lab:get-label ((labeled-source (σ x))|-r) = Some
(γ, n)
    unfolding p labeled-source-apply-subst[OF to-pterm-wf-pterm] using t p1 p
    by (smt (verit, ccfv-SIG) comp-apply fun-poss-imp-poss labeled-source-to-term
labelposs-obtain-label labelposs-subs-fun-poss poss-term-lab-to-term source-to-pterm
subset-eq eval-term.simps(1) subt-at-subst subterm-poss-conv)
  let ?p2=take (length r − n) r
  have ?p2 ≤p r by (metis append-take-drop-id less-eq-pos-simps(1))
  then have p1@?p2 ≤p p unfolding p by simp
  moreover have get-label ((labeled-source (σ x))|-?p2) = Some (γ, 0) ∧ ?p2 ∈
poss (labeled-source (σ x))
   using obtain-label-root[OF r-pos lab] well p1 x by (metis in-mono term.set-intros(3)
vars-term-subt-at)
  moreover then have ?p2 ∈ possL (σ x) using get-label-imp-labelposs by blast
```

ultimately show *?thesis* **using** *p1 x* **by** *blast*
qed

lemma *labelposs-subst*:
assumes $p \in$ *labelposs* $(t \cdot \sigma)$
shows $p \in$ *labelposs* $t \vee (\exists$ *p1 p2 x*. $p =$ *p1*@*p2* $\wedge$ *p1* $\in$ *poss* $t \wedge$ $t|$-*p1* $=$ *Var x* $\wedge$ *p2* $\in$ *labelposs* $(\sigma$ *x*$))$
using *assms* **proof**(*induct t arbitrary*:*p*)
case (*Fun fl ts*)
then show *?case* **proof**(*cases p*)
case *Nil*
from *Fun*(*2*) **obtain** *f l* **where** *fl* $=$ (*f*, *Some l*)
unfolding *eval-term.simps Nil* **by** (*metis get-label.simps*(*2*) *labelposs-obtain-label subt-at.simps*(*1*) *surjective-pairing*)
then show *?thesis*
unfolding *Nil* **by** *simp*
next
case (*Cons i p'*)
from *Fun*(*2*) **have** *i*:$i <$ *length ts*
unfolding *Cons eval-term.simps* **using** *labelposs-subs-poss* **by** *fastforce*
with *Fun*(*2*) **have** $p' \in$ *labelposs* $(ts!i \cdot \sigma)$
unfolding *Cons eval-term.simps* **by** (*metis* (*no-types*, *lifting*) *get-label-imp-labelposs labelposs-obtain-label labelposs-subs-poss nth-map option.simps*(*3*) *poss-Cons-poss subset-eq subt-at.simps*(*2*) *term.sel*(*4*))
with *Fun*(*1*) *i* **consider** $p' \in$ *labelposs* $(ts!i) \mid (\exists$ *p1 p2 x*. $p' =$ *p1* @ *p2* $\wedge$ *p1* $\in$ *poss* $(ts!i) \wedge (ts!i) \vdash$ *p1* $=$ *Var x* $\wedge$ *p2* $\in$ *labelposs* $(\sigma$ *x*$))$
by (*meson nth-mem*)
then show *?thesis* **proof**(*cases*)
case *1*
with *i* **show** *?thesis* **unfolding** *Cons*
by (*metis* (*no-types*, *lifting*) *get-label-imp-labelposs labelposs-obtain-label labelposs-subs-poss option.simps*(*3*) *poss-Cons-poss subsetD subt-at.simps*(*2*) *term.sel*(*4*))
next
case *2*
then obtain *p1 p2 x* **where** *p'*:$p' =$ *p1* @ *p2* **and** *p1*:*p1* $\in$ *poss* $(ts$! $i)$ *ts* ! $i \vdash$ *p1* $=$ *Var x* **and** *p2* $\in$ *labelposs* $(\sigma$ *x*$)$
by *blast*
with *i* **show** *?thesis* **unfolding** *Cons*
by (*metis append-Cons poss-Cons-poss subt-at.simps*(*2*) *term.sel*(*4*))
qed
qed
qed *simp*

lemma *set-labelposs-subst*:
labelposs $(t \cdot \sigma) =$ *labelposs* $t \cup (\bigcup i <$ *length* (*vars-term-list t*). $\{($*var-poss-list* $t!i)@q \mid q.\ q \in$ *labelposs* $(\sigma$ (*vars-term-list t* ! $i))\})$ (**is** *?ps* $=$ *?qs*)
proof
{fix *p* **assume** $p \in$ *?ps*
then consider $p \in$ *labelposs* $t \mid (\exists$ *p1 p2 x*. $p =$ *p1*@*p2* $\wedge$ *p1* $\in$ *poss* $t \wedge$ $t|$-*p1*

```
= Var x ∧ p2 ∈ labelpos (σ x))
      using labelpos-subst by blast
    then have p ∈ ?qs proof(cases)
      case 2
      then obtain p1 p2 x where p = p1@p2 ∧ p1 ∈ poss t ∧ t|-p1 = Var x ∧
p2 ∈ labelpos (σ x)
        by blast
     moreover then obtain i where i:i < length (vars-term-list t) vars-term-list
t ! i = x var-poss-list t ! i = p1
      by (metis in-set-idx length-var-poss-list term.inject(1) var-poss-iff var-poss-list-sound
vars-term-list-var-poss-list)
     ultimately have p ∈ {var-poss-list t ! i @ q |q. q ∈ labelpos (σ (vars-term-list
t ! i))}
        by blast
      with i(1) show ?thesis
        by blast
    qed simp
  }
  then show ?ps ⊆ ?qs
    by blast
  {fix q assume q ∈ ?qs
   then consider q ∈ labelpos t | q ∈ (⋃i< length (vars-term-list t). {(var-poss-list
t!i)@q | q. q ∈ labelpos (σ (vars-term-list t ! i))})
      by blast
    then have q ∈ ?ps proof(cases)
      case 1
      then show ?thesis proof(induct t arbitrary:q)
        case (Fun f ts)
        then show ?case proof(cases q)
          case Nil
          with Fun(2) obtain f' lab where f':f = (f', Some lab)
                by (metis get-label.simps(2) labelpos-obtain-label prod.exhaust-sel
subt-at.simps(1))
          show ?thesis unfolding Nil f' by simp
        next
          case (Cons i q')
          obtain f' lab where f':f = (f', lab)
            by fastforce
          show ?thesis proof(cases lab)
            case None
            from Fun(2) have i:i < length ts
              unfolding f' Cons None labelpos.simps by simp
            from Fun(2) have q' ∈ labelpos (ts!i)
              unfolding f' Cons None by simp
            with Fun(1) i have q' ∈ labelpos (ts!i · σ)
              by simp
            with i show ?thesis
              unfolding f' Cons None eval-term.simps labelpos.simps by simp
          next
```

case (*Some a*)
from *Fun*(*2*) **have** *i*:*i* < *length ts*
unfolding *f′ Cons Some labelposs.simps* **by** *simp*
from *Fun*(*2*) **have** *q′* ∈ *labelposs* (*ts*!*i*)
unfolding *f′ Cons Some* **by** *simp*
with *Fun*(*1*) *i* **have** *q′* ∈ *labelposs* (*ts*!*i* · *σ*)
by *simp*
with *i* **show** *?thesis*
unfolding *f′ Cons Some eval-term.simps labelposs.simps* **by** *simp*
qed
qed
qed *simp*
next
case *2*
then show *?thesis* **proof**(*induct t arbitrary*:*q*)
case (*Var x*)
have *var-poss-list* (*Var x*) = [[]]
unfolding *poss-list.simps var-poss.simps* **by** *simp*
with *Var* **show** *?case* **unfolding** *vars-term-list.simps*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def UN-iff bot-nat-0.not-eq-extremum length-0-conv length-nth-simps*(*2*) *lessThan-iff mem-Collect-eq not-less-eq nth-Cons-0 self-append-conv2 eval-term.simps*(*1*))
next
case (*Fun fl ts*)
from *Fun*(*2*) **obtain** *i q′* **where** *q*:*q* = *var-poss-list* (*Fun fl ts*) ! *i* @ *q′ q′* ∈ *labelposs* (*σ* (*vars-term-list* (*Fun fl ts*) ! *i*)) **and** *i*:*i* < *length* (*vars-term-list* (*Fun fl ts*))
by *blast*
then have *i′*:*i* < *length* (*var-poss-list* (*Fun fl ts*))
by (*metis length-var-poss-list*)
then obtain *j r* **where** *j*:*j* < *length ts var-poss-list* (*Fun fl ts*) ! *i* = *j*#*r*
unfolding *var-poss-list.simps* **by** (*smt* (*z3*) *add.left-neutral diff-zero length-map length-upt length-zip map-nth-eq-conv min.idem nth-concat-split nth-upt nth-zip prod.simps*(*2*))
with *i* **obtain** *x* **where** *x*:*Fun fl ts* |-(*j*#*r*) = *Var x*
by (*metis vars-term-list-var-poss-list*)
from *j i′* **have** *j*#*r* ∈ *var-poss* (*Fun fl ts*)
by (*metis nth-mem var-poss-list-sound*)
then have *r* ∈ *var-poss* (*ts*!*j*)
by *simp*
then obtain *i′* **where** *r*:*i′*<*length* (*var-poss-list* (*ts*!*j*)) *r* = *var-poss-list* (*ts*!*j*) ! *i′*
by (*metis in-set-conv-nth var-poss-list-sound*)
moreover then have (*vars-term-list* (*Fun fl ts*) ! *i*) = (*vars-term-list* (*ts*!*j*) ! *i′*)
using *x* **by** (*metis i j*(*2*) *length-var-poss-list subt-at.simps*(*2*) *term.inject*(*1*) *vars-term-list-var-poss-list*)
ultimately have *r*@*q′* ∈ (⋃ *i*<*length* (*vars-term-list* (*ts*!*j*)). {*var-poss-list* (*ts*!*j*) ! *i* @ *q* |*q*. *q* ∈ *labelposs* (*σ* (*vars-term-list* (*ts*!*j*) ! *i*))})

using *q*(*2*) **unfolding** *length-var-poss-list* **by** *auto*
with *Fun*(*1*) *j*(*1*) **have** *r-pos*:*r*@*q'* ∈ *labelposs* ((*ts*!*j*) · *σ*)
using *nth-mem* **by** *blast*
obtain *f lab* **where** *f*:*fl* = (*f*, *lab*)
using *surjective-pairing* **by** *blast*
then show *?case* **proof**(*cases lab*)
case *None*
from *r-pos* **have** *j*#*r*@*q'* ∈ *labelposs* (*Fun fl ts* · *σ*)
unfolding *eval-term.simps f None labelposs.simps length-map* **using** *j*(*1*)
by *simp*
then show *?thesis* **unfolding** *q j*(*2*) **by** *simp*
next
case (*Some a*)
from *r-pos* **have** *j*#*r*@*q'* ∈ *labelposs* (*Fun fl ts* · *σ*)
unfolding *eval-term.simps f Some labelposs.simps length-map* **using** *j*(*1*)
by *simp*
then show *?thesis* **unfolding** *q j*(*2*) **by** *simp*
qed
qed
qed
}
then show *?qs* ⊆ *?ps*
by *blast*
qed

The labeled positions in a proof term *Prule α As* are the function positions of *lhs α* together with all labeled positions in the arguments *As*.

lemma *possl-rule*:
assumes *length As* = *length* (*var-rule α*) *linear-term* (*lhs α*)
shows *possL* (*Prule α As*) = *fun-poss* (*lhs α*) ∪ (⋃*i*< (*length As*). {(*var-poss-list* (*lhs α*)!*i*)@*q* | *q*. *q* ∈ *possL*(*As*!*i*)})
proof−
from *assms*(*1*,*2*) **have** *l*:*length* (*vars-term-list* (*labeled-lhs α*)) = *length As*
by (*metis linear-term-var-vars-term-list vars-term-list-labeled-lhs*)
have *labelposs* (*labeled-lhs α*) = *fun-poss* (*lhs α*)
by (*metis fun-poss-term-lab-to-term label-poss-labeled-lhs label-term-to-term labelposs-subs-fun-poss subsetI subset-antisym*)
moreover from *assms*(*1*,*2*) **have** *i* < *length As* ⟹ (⟨*map labeled-source As*⟩$_\alpha$) (*vars-term-list* (*labeled-lhs α*) ! *i*) = *labeled-source* (*As*!*i*) **for** *i*
using *lhs-subst-var-i linear-term-var-vars-term-list* **by** (*smt* (*verit*, *best*) *length-map nth-map vars-term-list-labeled-lhs*)
ultimately show *?thesis* **using** *set-labelposs-subst*[*of labeled-lhs α*] **unfolding** *l var-poss-list-labeled-lhs* **by** *force*
qed

lemma *labelposs-subs-fun-poss-source*:
assumes *p* ∈ *possL A*
shows *p* ∈ *fun-poss* (*source A*)
proof−

have $p \in$ *fun-poss* (*labeled-source A*)
using *assms labelposs-subs-fun-poss* **by** *blast*
then show *?thesis* **using** *fun-poss-term-lab-to-term*
by *auto*
qed

The labeled source of a context (obtained from some proof term A) applied to some proof term B is the labeled source of the context applied to the labeled source of the proof term B.

context *left-lin*
begin
lemma *label-source-ctxt*:
assumes $A \in$ *wf-pterm R*
and *ctxt-of-pos-term p* (*source A*) = *source-ctxt* (*ctxt-of-pos-term p′ A*)
and $p \in$ *poss* (*source A*) **and** $p' \in$ *poss A*
shows *labeled-source* (*ctxt-of-pos-term p′ A*)⟨*B*⟩ = (*ctxt-of-pos-term p* (*labeled-source A*))⟨*labeled-source B*⟩
using *assms* **proof**(*induct p′ arbitrary*:*p A*)
case *Nil*
then have *p*:*p* = []
using *hole-pos-ctxt-of-pos-term* **by** *force*
then show *?case* **by** *simp*
next
case (*Cons i p′*)
then obtain *fl As* **where** *a*:*A* = *Fun fl As* **and** *i*:*i* < *length As* **and** *p′*:*p′* ∈ *poss* (*As*!*i*)
by (*meson args-poss*)
then show *?case* **proof**(*cases fl*)
case (*Inl α*)
from *Cons*(*2*) **have** *l*:*length As* = *length* (*var-rule α*)
unfolding *a Inl* **using** *wf-pterm.cases* **by** *auto*
have *to-rule α* ∈ *R*
using *Cons*(*2*) **unfolding** *a Inl* **using** *wf-pterm.cases* **by** *force*
with *left-lin* **have** *lin*:*linear-term* (*lhs α*)
using *left-linear-trs-def* **by** *fastforce*
let *?p1*=*var-poss-list* (*lhs α*) ! *i*
from *i l lin* **have** *p1*:(*lhs α*)|-*?p1* = *Var* (*var-rule α* ! *i*)
by (*metis linear-term-var-vars-term-list vars-term-list-var-poss-list*)
from *i l* **have** *p1-pos*:*?p1* ∈ *poss* (*lhs α*)
by (*metis comp-apply length-remdups-leq length-rev length-var-poss-list nth-mem order-less-le-trans var-poss-imp-poss var-poss-list-sound*)
let *?p2*=*hole-pos* (*source-ctxt* (*ctxt-of-pos-term p′* (*As* ! *i*)))
have *hole-pos* (*source-ctxt* (*ctxt-of-pos-term* (*i* # *p′*) *A*)) = *?p1*@*?p2*
unfolding *a Inl source.simps ctxt-of-pos-term.simps source-ctxt.simps Let-def hole-pos-ctxt-compose* **using** *p1-pos Cons*(*5*) *a* **by** *force*
with *Cons*(*3*) **have** *p*:*p* = *?p1*@*?p2*
by (*metis Cons.prems*(*3*) *hole-pos-ctxt-of-pos-term*)
have *at-p1*:(*source A*)|-*?p1* = *source* (*As*!*i*)
unfolding *a Inl source.simps* **using** *p1*

by (*smt* (*verit*, *best*) *Inl i l length-map lhs-subst-var-i nth-map p1-pos eval-term.simps*(*1*) *subt-at-subst*)
with *Cons*(*4*) **have** *p2-pos*:*?p2* ∈ *poss* (*source* (*As*!*i*))
unfolding *p* **by** *simp*
from *at-p1* **have** ∗:*ctxt-of-pos-term p* (*source A*) = (*ctxt-of-pos-term ?p1* (*source A*) ∘$_c$ (*ctxt-of-pos-term ?p2* (*source* (*As* ! *i*))))
unfolding *p* **using** *ctxt-of-pos-term-append* **using** *Cons.prems*(*3*) *p* **by** *fastforce*
from *Cons*(*3*) **have** *ctxt-of-pos-term ?p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term p′* (*As*!*i*))
unfolding ∗ **unfolding** *a Inl source.simps ctxt-of-pos-term.simps source-ctxt.simps Let-def* **using** *ctxt-comp-equals Cons*(*5*) *p1-pos*
by (*smt* (*verit*, *ccfv-SIG*) *a ctxt-of-pos-term.simps*(*2*) *hole-pos.simps*(*2*) *hole-pos-ctxt-of-pos-term list.inject poss-imp-subst-poss*)
with *Cons*(*1*,*2*) *i p2-pos p′ a Inl* **have** *IH*:*labeled-source* (*ctxt-of-pos-term p′* (*As*!*i*))⟨*B*⟩ = (*ctxt-of-pos-term ?p2* (*labeled-source* (*As*!*i*)))⟨*labeled-source B*⟩
by (*meson fun-well-arg nth-mem*)
then have *list-IH*:*map labeled-source* (*take i As* @ (*ctxt-of-pos-term p′* (*As* ! *i*))⟨*B*⟩ # *drop* (*Suc i*) *As*) =
map labeled-source (*take i As*) @ (*ctxt-of-pos-term ?p2* (*labeled-source* (*As* ! *i*)))⟨*labeled-source B*⟩ # *map labeled-source* (*drop* (*Suc i*) *As*)
using *i* **by** *fastforce*
from *lin* **have** *lin′*:*linear-term* (*labeled-lhs* α)
using *linear-label-term* **by** *blast*
from *p1-pos* **have** *p1-pos*:*?p1* ∈ *poss* (*labeled-lhs* α)
by *simp*
from *p1* **have** *x*:*labeled-lhs* α |- *var-poss-list* (*lhs* α) ! *i* = *Var* (*var-rule* α ! *i*)
by (*metis label-term-to-term p1-pos poss-term-lab-to-term var-label-term*)
have (⟨*map labeled-source As*⟩$_\alpha$)((*var-rule* α ! *i*) := (*ctxt-of-pos-term ?p2* ((⟨*map labeled-source As*⟩$_\alpha$) (*var-rule* α ! *i*)))⟨*labeled-source B*⟩)
= ⟨(*take i* (*map labeled-source As*)) @ (*ctxt-of-pos-term ?p2* (*labeled-source* (*As*!*i*)))⟨*labeled-source B*⟩ # (*drop* (*Suc i*) (*map labeled-source As*))⟩$_\alpha$
using *i* **by** (*smt* (*verit*, *best*) *Cons.prems*(*4*) *a ctxt-of-pos-term.simps*(*2*) *hole-pos.simps*(*2*) *hole-pos-ctxt-of-pos-term id-take-nth-drop l length-map lhs-subst-upd lhs-subst-var-i list.inject nth-map take-map*)
then show *?thesis* **unfolding** *a Inl ctxt-of-pos-term.simps labeled-source.simps intp-actxt.simps p list-IH*
using *replace-at-append-subst*[*OF lin′ p1-pos x*] **by** (*smt* (*verit*) *drop-map take-map*)
next
case (*Inr f*)
from *Cons*(*3*,*4*,*5*) **obtain** *p2* **where** *p*:*p* = *i*#*p2* **and** *p2*:*p2* ∈ *poss* (*source* (*As*!*i*)) **and** *ctxt*:*ctxt-of-pos-term p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term p′* (*As*!*i*))
unfolding *a Inr source.simps ctxt-of-pos-term.simps source-ctxt.simps* **by** (*smt* (*verit*, *best*) *Cons.prems*(*2*) *Cons.prems*(*3*) *Inr a actxt.inject ctxt-of-pos-term.simps*(*2*) *i nth-map source-poss*)
from *Cons*(*1*,*2*) *ctxt p2 p′* **have** *IH*:*labeled-source* (*ctxt-of-pos-term p′* (*As*!*i*))⟨*B*⟩ = (*ctxt-of-pos-term p2* (*labeled-source* (*As*!*i*)))⟨*labeled-source B*⟩

```
    using a i nth-mem by blast
   then have list-IH:map labeled-source (take i As @ (ctxt-of-pos-term p' (As !
i))⟨B⟩ # drop (Suc i) As) =
     map labeled-source (take i As) @ (ctxt-of-pos-term p2 (labeled-source (As !
i)))⟨labeled-source B⟩ # map labeled-source (drop (Suc i) As)
    using i by fastforce
   show ?thesis unfolding a Inr ctxt-of-pos-term.simps p labeled-source.simps
intp-actxt.simps list-IH
    by (simp add: drop-map i take-map)
 qed
qed
end

lemma labeled-ctxt-above:
 assumes p ∈ poss A and r ∈ poss A and ¬ p ≤p r
 shows get-label ((ctxt-of-pos-term p A)⟨labeled-source B⟩ |-r) = get-label (A |-r)
using assms proof(induct A arbitrary:r p)
 case (Fun f As)
 then have p ≠ []
  by blast
 with Fun(2) obtain i p' where i:i < length As and p':p' ∈ poss (As!i) and p:p
= i#p'
  by auto
 from Fun(4) consider r <p p | r ⊥ p
  using parallel-pos by fastforce
 then show ?case proof(cases)
  case 1
  then show ?thesis proof(cases r)
   case Nil
   show ?thesis unfolding p Nil by simp
  next
   case (Cons j r')
   from 1 have j:j = i
    unfolding p Cons by simp
   with Fun(1) have get-label ((ctxt-of-pos-term p' (As!i))⟨labeled-source B⟩ |-
r') = get-label ((As!i) |- r')
    using i p' Fun(3,4) unfolding Cons j p by simp
   then show ?thesis
    unfolding Cons p subt-at.simps ctxt-of-pos-term.simps intp-actxt.simps by
(metis i j nat-less-le nth-append-take)
  qed
 next
  case 2
  then obtain j r' where r:r = j#r'
   unfolding p
   by (metis Nil-not-par parallel-pos-sym proper-prefix-list.cases)
  then show ?thesis proof(cases i = j)
   case True
   from Fun(1) 2 i have get-label ((ctxt-of-pos-term p' (As!i))⟨labeled-source B⟩
```

```
|- r') = get-label ((As!i) |- r')
        using Fun.prems(2) Fun.prems(3) True p p' r by force
      then show ?thesis using p r True
      by (metis 2 Fun.prems(1) Fun.prems(2) parallel-pos parallel-replace-at-subt-at)

    next
      case False
      then show ?thesis
          unfolding p r subt-at.simps ctxt-of-pos-term.simps intp-actxt.simps by
(metis i nth-list-update upd-conv-take-nth-drop)
    qed
  qed
qed simp
```

The labeled positions of a context (obtained from some proof term A) applied to some proof term B are the labeled positions of the context together with the labeled positions of the proof term B.

```
context left-lin
begin
lemma label-ctxt:
  assumes A ∈ wf-pterm R
  and ctxt-of-pos-term p (source A) = source-ctxt (ctxt-of-pos-term p' A)
  and p ∈ poss (source A) and p' ∈ poss A
  shows possL (ctxt-of-pos-term p' A)⟨B⟩ = {q. q ∈ possL A ∧ ¬p ≤p q} ∪ {p@q|
q. q ∈ possL B}
  using assms proof(induct p' arbitrary:p A)
  case Nil
  then have p:p = []
    using hole-pos-ctxt-of-pos-term by force
  then have {q ∈ possL A. ¬ p ≤p q} = {}
    by simp
  then show ?case
    unfolding Nil ctxt-of-pos-term.simps p by simp
next
  case (Cons i p')
  then obtain fl As where a:A = Fun fl As and i:i < length As and p':p' ∈ poss
(As!i)
    by (meson args-poss)
  then show ?case proof(cases fl)
    case (Inl α)
    from Cons(2) have l:length As = length (var-rule α)
      unfolding a Inl using wf-pterm.cases by auto
    have to-rule α ∈ R
      using Cons(2) unfolding a Inl using wf-pterm.cases by force
    with left-lin have lin:linear-term (lhs α)
      using left-linear-trs-def by fastforce
    let ?p1=var-poss-list (lhs α) ! i
    from i l lin have p1:(lhs α)|-?p1 = Var (var-rule α ! i)
      by (metis linear-term-var-vars-term-list vars-term-list-var-poss-list)
```

from *i l* **have** *p1-pos*:*?p1* ∈ *poss* (*lhs* α)
by (*metis comp-apply length-remdups-leq length-rev length-var-poss-list nth-mem order-less-le-trans var-poss-imp-poss var-poss-list-sound*)
let *?p2*=*hole-pos* (*source-ctxt* (*ctxt-of-pos-term p′* (*As* ! *i*)))
have *hole-pos* (*source-ctxt* (*ctxt-of-pos-term* (*i* # *p′*) *A*)) = *?p1*@*?p2*
unfolding *a Inl source.simps ctxt-of-pos-term.simps source-ctxt.simps Let-def hole-pos-ctxt-compose* **using** *p1-pos Cons*(*5*) *a* **by** *force*
with *Cons*(*3*) **have** *p*:*p* = *?p1*@*?p2*
by (*metis Cons.prems*(*3*) *hole-pos-ctxt-of-pos-term*)
have *at-p1*:(*source A*)|-*?p1* = *source* (*As*!*i*)
unfolding *a Inl source.simps* **using** *p1*
by (*smt* (*verit*, *best*) *Inl i l length-map lhs-subst-var-i nth-map p1-pos eval-term.simps*(*1*) *subt-at-subst*)
with *Cons*(*4*) **have** *p2-pos*:*?p2* ∈ *poss* (*source* (*As*!*i*))
unfolding *p* **by** *simp*
from *at-p1* **have** ∗:*ctxt-of-pos-term p* (*source A*) = (*ctxt-of-pos-term ?p1* (*source A*) ∘$_c$ (*ctxt-of-pos-term ?p2* (*source* (*As* ! *i*))))
unfolding *p* **using** *ctxt-of-pos-term-append* **using** *Cons.prems*(*3*) *p* **by** *fast-force*
from *Cons*(*3*) **have** *ctxt-of-pos-term ?p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term p′* (*As*!*i*))
unfolding ∗ **unfolding** *a Inl source.simps ctxt-of-pos-term.simps source-ctxt.simps Let-def* **using** *ctxt-comp-equals Cons*(*5*) *p1-pos*
by (*smt* (*verit*, *ccfv-SIG*) *a ctxt-of-pos-term.simps*(*2*) *hole-pos.simps*(*2*) *hole-pos-ctxt-of-pos-term list.inject poss-imp-subst-poss*)
with *Cons*(*1*,*2*) *i p2-pos p′ a Inl* **have** *IH*:*possL* (*ctxt-of-pos-term p′* (*As*!*i*))⟨*B*⟩ = {*q* ∈ *possL* (*As*!*i*). ¬ *?p2* ≤$_p$ *q*} ∪ {*?p2* @ *q* |*q*. *q* ∈ *possL B*}
by (*meson fun-well-arg nth-mem*)
let *?a1*=*fun-poss* (*lhs* α)
let *?a2*=(⋃*j* ∈ {*k*. *k* < *length As* ∧ *k* ≠ *i*}. {(*var-poss-list* (*lhs* α)!*j*)@*q* | *q*. *q* ∈ *possL*(*As*!*j*)})
let *?a3*={*?p1*@*q* | *q*. *q* ∈ *possL* (*As*!*i*) ∧ ¬ *?p2* ≤$_p$ *q*}
let *?a4*={*?p1* @ *?p2* @ *q* |*q*. *q* ∈ *possL B*}
let *?b1*={*q* ∈ *possL A*. ¬ *p* ≤$_p$ *q*}
have *?a1* ∪ *?a2* ∪ *?a3* = *?b1* **proof**
{fix *x* **assume** *x*:*x* ∈ *?a1*
then have ¬ *?p1* ≤$_p$ *x*
by (*metis append.right-neutral fun-poss-append-poss fun-poss-fun-conv fun-poss-imp-poss p1 prefix-pos-diff term.distinct*(*1*))
then have ¬ *p* ≤$_p$ *x*
unfolding *p* **using** *less-eq-pos-simps*(*1*) *prefix-order.order-trans* **by** *blast*
with *x* **have** *x* ∈ *?b1*
unfolding *a Inl* **using** *possl-rule l lin* **by** *auto*
} moreover {fix *x* **assume** *x* ∈ *?a2*
then obtain *j q* **where** *j*:*j* < *length As j* ≠ *i* **and** *q*:*q* ∈ *possL* (*As* ! *j*) **and** *x*:*x* = *var-poss-list* (*lhs* α) ! *j* @ *q*
by *blast*
from *j* **have** *j′*:*j* < *length* (*var-poss-list* (*lhs* α))
using *l lin* **by** (*metis length-var-poss-list linear-term-var-vars-term-list*)

with *j*(*2*) **have** *?p1* ≠ (*var-poss-list* (*lhs* α)) !*j*
by (*metis* (*mono-tags*, *lifting*) *distinct-remdups distinct-rev i j*(*1*) *l lin lin-ear-term-var-vars-term-list nth-eq-iff-index-eq o-apply term.inject*(*1*) *vars-term-list-var-poss-list*)
with *j′* **have** *?p1* ⊥ *var-poss-list* (*lhs* α) ! *j*
using *var-poss-parallel* **by** (*metis nth-mem p1 p1-pos var-poss-iff var-poss-list-sound*)
then have ¬ *p* ≤$_p$ *x*
unfolding *p x* **using** *less-eq-pos-simps*(*1*) *prefix-order.order-trans pos-less-eq-append-not-parallel* **by** *blast*
then have *x* ∈ *?b1*
unfolding *a Inl possl-rule*[*OF l lin*] *x* **using** *j*(*1*) *q* **by** *blast*
} **moreover** {**fix** *x* **assume** *x* ∈ *?a3*
then obtain *q* **where** *x*:*x* = *?p1*@*q q* ∈ *possL* (*As* ! *i*) ¬ *?p2* ≤$_p$ *q*
by *blast*
from *x*(*3*) **have** ¬ *p* ≤$_p$ *x*
unfolding *p x*(*1*) **using** *less-eq-pos-simps*(*2*) **by** *blast*
with *x*(*2*) **have** *x* ∈ *?b1*
unfolding *a Inl possl-rule*[*OF l lin*] *x*(*1*) **using** *i* **by** *auto*
}
ultimately show *?a1* ∪ *?a2* ∪ *?a3* ⊆ *?b1* **by** *blast*
{**fix** *x* **assume** *b1*:*x* ∈ *?b1*
then consider *x* ∈ *fun-poss* (*lhs* α) | *x* ∈ (⋃ *i*<*length As*. {*var-poss-list* (*lhs* α) ! *i* @ *q* |*q*. *q* ∈ *possL* (*As* ! *i*)})
unfolding *a Inl possl-rule*[*OF l lin*] **by** *blast*
then have *x* ∈ *?a1* ∪ *?a2* ∪ *?a3* **proof**(*cases*)
case *2*
then obtain *j q* **where** *j*:*j* < *length As* **and** *x*:*x* = *var-poss-list* (*lhs* α) ! *j* @ *q* **and** *q*:*q* ∈ *possL* (*As*!*j*)
by *blast*
then show *?thesis* **proof**(*cases j* = *i*)
case *True*
from *b1* **have** ¬ *?p2* ≤$_p$ *q*
unfolding *p x True* **using** *less-eq-pos-simps*(*2*) **by** *blast*
then show *?thesis* **using** *j x q* **by** *auto*
qed *auto*
qed *simp*
}
then show *?b1* ⊆ *?a1* ∪ *?a2* ∪ *?a3* **by** *blast*
qed
moreover have *possL* (*ctxt-of-pos-term* (*i* # *p′*) *A*)⟨*B*⟩ = *?a1* ∪ *?a2* ∪ *?a3* ∪ *?a4* **proof**−
from *l i* **have** *l′*:*length* (*take i As* @ (*ctxt-of-pos-term p′* (*As* ! *i*))⟨*B*⟩ # *drop* (*Suc i*) *As*) = *length* (*var-rule* α)
by *simp*
have *set*:{*j*. *j* < *length As*} = {*j*. *j* < *length As* ∧ *j* ≠ *i*} ∪ {*i*}
using *i Collect-disj-eq* **by** *auto*
let *?args*=(*take i As* @ (*ctxt-of-pos-term p′* (*As* ! *i*))⟨*B*⟩ # *drop* (*Suc i*) *As*)
{**fix** *j* **assume** *j* < *length As* ∧ *j* ≠ *i*
with *i* **have** *?args* ! *j* = *As*!*j*
by (*meson nat-less-le nth-append-take-drop-is-nth-conv*)

} **moreover have** *?args!i = (ctxt-of-pos-term p′ (As ! i))⟨B⟩* **using** *i*
by (*simp add*: *nth-append-take*)
moreover from *set* **have** (⋃*j*<*length As*. {*var-poss-list* (*lhs α*) ! *j* @ *q* |*q*. *q* ∈ *possL* (*?args* ! *j*)}) =
(⋃*j* ∈ {*j*. *j* < *length As* ∧ *j* ≠ *i*}. {*var-poss-list* (*lhs α*) ! *j* @ *q* |*q*. *q* ∈ *possL* (*?args* ! *j*)}) ∪ {*?p1* @ *q* |*q*. *q* ∈ *possL* (*?args!i*)}
by *force*
ultimately have (⋃*j*<*length As*. {*var-poss-list* (*lhs α*) ! *j* @ *q* |*q*. *q* ∈ *possL* (*?args* ! *j*)}) =
(⋃*j* ∈ {*j*. *j* < *length As* ∧ *j* ≠ *i*}. {*var-poss-list* (*lhs α*) ! *j* @ *q* |*q*. *q* ∈ *possL* (*As* ! *j*)}) ∪ {*?p1* @ *q* |*q*. *q* ∈ *possL* (*ctxt-of-pos-term p′* (*As* ! *i*))⟨*B*⟩}
by *simp*
moreover have *possL* (*ctxt-of-pos-term* (*i* # *p′*) *A*)⟨*B*⟩ = *fun-poss* (*lhs α*) ∪
(⋃*j*<*length As*. {*var-poss-list* (*lhs α*) ! *j* @ *q* |*q*. *q* ∈ *possL* (*?args* ! *j*)})
unfolding *a Inl ctxt-of-pos-term.simps intp-actxt.simps* **using** *possl-rule*[*OF l′ lin*] *i* **by** *force*
ultimately show *?thesis* **unfolding** *IH* **by** *auto*
qed
ultimately show *?thesis* **using** *p* **by** *force*
next
case (*Inr f*)
from *Cons*(*3*,*4*,*5*) **obtain** *p2* **where** *p*:*p* = *i*#*p2* **and** *p2* ∈ *poss* (*source* (*As*!*i*)) **and** *ctxt*:*ctxt-of-pos-term p2* (*source* (*As*!*i*)) = *source-ctxt* (*ctxt-of-pos-term p′* (*As*!*i*))
unfolding *a Inr source.simps ctxt-of-pos-term.simps source-ctxt.simps* **by** (*smt* (*verit*, *best*) *Cons.prems*(*2*) *Cons.prems*(*3*) *Inr a actxt.inject ctxt-of-pos-term.simps*(*2*) *i nth-map source-poss*)
with *Cons*(*1*,*2*) *i p′* **have** *IH*:*possL* (*ctxt-of-pos-term p′* (*As*!*i*))⟨*B*⟩ = {*q* ∈ *possL* (*As*!*i*). ¬ *p2* ≤*p q*} ∪ {*p2* @ *q* |*q*. *q* ∈ *possL B*}
unfolding *a Inr* **by** (*meson fun-well-arg nth-mem*)
let *?a2*=(⋃*j* ∈ {*k*. *k* < *length As* ∧ *k* ≠ *i*}. {*j* # *q* | *q*. *q* ∈ *possL*(*As*!*j*)})
let *?a3*={*i*#*q* | *q*. *q* ∈ *possL* (*As*!*i*) ∧ ¬ *p2* ≤*p q*}
let *?a4*={*i* # *p2* @ *q* |*q*. *q* ∈ *possL B*}
let *?b1*={*q* ∈ *possL A*. ¬ *p* ≤*p q*}
have *?a2* ∪ *?a3* = *?b1* **proof**
{fix *x* **assume** *x* ∈ *?a2*
then obtain *j q* **where** *j*:*j* < *length As j* ≠ *i* **and** *q*:*q* ∈ *possL* (*As* ! *j*) **and** *x*:*x* = *j* # *q*
by *blast*
from *j q* **have** *j*#*q* ∈ *possL A*
unfolding *a Inr* **by** *simp*
then have *x* ∈ *?b1*
unfolding *x p* **using** *j*(*2*) **by** *simp*
} **moreover {fix** *x* **assume** *x* ∈ *?a3*
then obtain *q* **where** *x*:*x* = *i*#*q q* ∈ *possL* (*As* ! *i*) ¬ *p2* ≤*p q*
by *blast*
from *x*(*3*) **have** ¬ *p* ≤*p x*
unfolding *p x*(*1*) **using** *less-eq-pos-simps*(*2*) **by** *simp*

```
      with x(2) have x ∈ ?b1
        unfolding a Inr x(1) using i by auto
     }
     ultimately show ?a2 ∪ ?a3 ⊆ ?b1 by blast
     {fix x assume b1:x ∈ ?b1
      then have x ∈ possL A
        by simp
      then obtain j q where j:j < length As and x:x = j # q and q:q ∈ possL
(As!j)
        unfolding a Inr labeled-source.simps labelposs.simps length-map by force
      then have x ∈ ?a2 ∪ ?a3 proof(cases j = i)
        case True
        with b1 have ¬ p2 ≤p q
          unfolding p x using less-eq-pos-simps(2) by simp
        then show ?thesis using j x q b1 by auto
      qed simp
     }
     then show ?b1 ⊆ ?a2 ∪ ?a3 by blast
   qed
   moreover have possL (ctxt-of-pos-term (i # p') A)⟨B⟩ = ?a2 ∪ ?a3 ∪ ?a4
proof−
     have l:length (take i As @ (ctxt-of-pos-term p' (As ! i))⟨B⟩ # drop (Suc i)
As) = length As
       using i by simp
     {fix j assume j < length As
      then have (map labeled-source (take i As @ (ctxt-of-pos-term p' (As ! i))⟨B⟩
# drop (Suc i) As) ! j) = labeled-source ((take i As @ (ctxt-of-pos-term p' (As !
i))⟨B⟩ # drop (Suc i) As) ! j)
        using nth-map l by metis
     }note map-lab=this
     have set:{j. j < length As} = {j. j < length As ∧ j ≠ i} ∪ {i}
       using i Collect-disj-eq by auto
     let ?args=(take i As @ (ctxt-of-pos-term p' (As ! i))⟨B⟩ # drop (Suc i) As)
     {fix j assume j < length As ∧ j ≠ i
       with i have ?args ! j = As!j
         by (meson nat-less-le nth-append-take-drop-is-nth-conv)
     } moreover have ?args!i = (ctxt-of-pos-term p' (As ! i))⟨B⟩ using i
       by (simp add: nth-append-take)
     moreover from set have (⋃j<length As. {j # q |q. q ∈ possL (?args ! j)})
=
                  (⋃j ∈ {j. j < length As ∧ j ≠ i}. {j # q |q. q ∈ possL (?args !
j)}) ∪ {i # q |q. q ∈ possL (?args!i)}
       by force
     ultimately have (⋃j<length As. {j # q |q. q ∈ possL (?args ! j)}) =
            (⋃j ∈ {j. j < length As ∧ j ≠ i}. {j # q |q. q ∈ possL (As ! j)}) ∪
{i # q |q. q ∈ possL (ctxt-of-pos-term p' (As ! i))⟨B⟩}
       by simp
    moreover have possL (ctxt-of-pos-term (i # p') A)⟨B⟩ = (⋃j<length As. {j
# q |q. q ∈ possL (?args ! j)})
```

unfolding *a Inr ctxt-of-pos-term.simps intp-actxt.simps labeled-source.simps labelposs.simps length-map l* **using** *map-lab* **by** *force*
ultimately show *?thesis* **unfolding** *IH* **by** *auto*
qed
ultimately show *?thesis* **using** *p* **by** *force*
qed
qed

lemma *single-redex-possL*:
assumes *to-rule* $\alpha \in R$ $p \in poss\ s$
shows *possL* (*ll-single-redex s p* α) = $\{p$ @ q |*q*. $q \in$ *fun-poss* (*lhs* α)$\}$
proof−
let *?*Δ=*ll-single-redex s p* α
have ∗:*possL* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α)))) = *labelposs* (*labeled-lhs* α)
proof−
{fix *x*
have *labelposs* ((⟨*map labeled-source* (*map* (*to-pterm* ∘ (λ*pi*. *s* |- (*p* @ *pi*))) (*var-poss-list* (*lhs* α)))⟩$_\alpha$) *x*) = {}
by (*smt* (*verit*) *comp-apply labeled-source-simple-pterm labelposs.simps*(*1*) *length-map lhs-subst-not-var-i lhs-subst-var-i map-nth-eq-conv*)
}
then show *?thesis* **unfolding** *labeled-source.simps labelposs-apply-subst* **by** *blast*
qed
have *possL ?*Δ = $\{q \in$ *possL* (*to-pterm s*). ¬ $p \leq_p q\} \cup \{p$ @ q |*q*. $q \in$ *possL* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α))))$\}$
using *label-ctxt assms* **by** (*simp add*: *ll-single-redex-def p-in-poss-to-pterm source-ctxt-to-pterm*)
also have ...= $\{p$ @ q |*q*. $q \in$ *possL* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α))))$\}$
using *labeled-source-simple-pterm* **by** *auto*
also have ...= $\{p$ @ q |*q*. $q \in$ *labelposs* (*labeled-lhs* α)$\}$
unfolding ∗ **by** *simp*
finally show *?thesis*
using *label-poss-labeled-lhs labelposs-subs-fun-poss* **by** *fastforce*
qed

end

lemma *labeled-poss-in-lhs*:
assumes *p-pos*:*p* ∈ *poss* (*source* (*Prule* α *As*)) **and** *well*:*Prule* α *As* ∈ *wf-pterm R*
and *get-label* ((*labeled-source* (*Prule* α *As*))|-*p*) = *Some* (α, *length p*) *is-Fun* (*lhs* α)
shows $p \in$ *fun-poss* (*lhs* α)
proof−
from *p-pos* **consider** $p \in$ *fun-poss* (*lhs* α) | ∃ *p1 p2 x*. p = *p1* @ *p2* ∧ *p1* ∈ *poss*

$(lhs\ \alpha) \wedge (lhs\ \alpha)|\text{-}p1 = Var\ x \wedge p2 \in poss\ ((\langle map\ source\ As\rangle_\alpha)\ x)$
unfolding *source.simps* **using** *poss-subst-apply-term* **by** *metis*
then show *?thesis* **proof**(*cases*)
case *2*
then obtain *p1 p2 x* **where** *p*:$p = p1\ @\ p2$ **and** *p1*:$p1 \in poss\ (lhs\ \alpha)\ (lhs\ \alpha)|\text{-}p1 = Var\ x$ **and** *p2*:$p2 \in poss\ ((\langle map\ source\ As\rangle_\alpha)\ x)$
by *blast*
then obtain *i* **where** *i*:$i < length\ (var\text{-}rule\ \alpha)$ $var\text{-}rule\ \alpha!i = x$
by (*metis in-set-conv-nth set-vars-term-list subt-at-imp-supteq subteq-Var-imp-in-vars-term vars-term-list-vars-distinct*)
from *p1* **have** *p1-pos'*:$p1 \in poss\ (labeled\text{-}lhs\ \alpha)$
by *simp*
from *p1* **have** *p1-pos*:$p1 \in poss\ (labeled\text{-}lhs\ \alpha \cdot \langle map\ labeled\text{-}source\ As\rangle_\alpha)$
by (*metis labeled-source.simps*(*3*) *labeled-source-to-term p p-pos poss-append-poss poss-term-lab-to-term*)
from *p1* **have** *x*:$labeled\text{-}lhs\ \alpha\ |\text{-}p1 = Var\ x$
by (*metis fun-poss-term-lab-to-term label-term-to-term labeled-source-pos poss-simps*(*4*) *poss-term-lab-to-term term.sel*(*1*) *term-lab-to-term.simps*(*1*) *var-poss-iff*)
from *well* **have** *l*:$length\ As = length\ (var\text{-}rule\ \alpha)$
using *wf-pterm.cases* **by** *auto*
with *well i* **have** *asi*:$As!i \in wf\text{-}pterm\ R$
by (*metis fun-well-arg nth-mem*)
from *l* **have** *lab*:$labeled\text{-}source\ (Prule\ \alpha\ As)\ |\text{-}p = labeled\text{-}source\ (As!i)\ |\text{-}p2$
unfolding *p labeled-source.simps subt-at-append*[*OF p1-pos*] *subt-at-subst*[*OF p1-pos'*] *x* **using** *i*
by (*metis* (*no-types*, *lifting*) *length-map lhs-subst-var-i nth-map eval-term.simps*(*1*))
moreover from *assms*(*4*) *p1* **have** $length\ p2 < length\ p$
unfolding *p* **by** *auto*
moreover from *p2* **have** $p2 \in poss\ (labeled\text{-}source\ (As!i))$
using *l i* **by** (*metis* (*no-types*, *lifting*) *labeled-source-to-term length-map lhs-subst-var-i nth-map poss-term-lab-to-term*)
ultimately have *False* **using** *assms*(*3*) *asi* **by** (*simp add*: *label-term-max-value leD*)
then show *?thesis* **by** *simp*
qed *simp*
qed

context *left-lin-no-var-lhs*
begin
lemma *get-label-Prule*:
assumes $Prule\ \alpha\ As \in wf\text{-}pterm\ R$ **and** $p \in poss\ (source\ (Prule\ \alpha\ As))$ **and** $get\text{-}label\ (labeled\text{-}source\ (Prule\ \alpha\ As)\ |\text{-}\ p) = Some\ (\beta,\ 0)$
shows $(p = [] \wedge \alpha = \beta) \vee$
$(\exists\ p1\ p2\ i.\ p = p1@p2 \wedge i < length\ As \wedge var\text{-}poss\text{-}list\ (lhs\ \alpha)!i = p1 \wedge$
$p2 \in poss\ (source\ (As!i)) \wedge get\text{-}label\ (labeled\text{-}source\ (As!i)|\text{-}p2) = Some\ (\beta,\ 0))$
proof−
from *assms*(*1*) **have** α:$to\text{-}rule\ \alpha \in R$
using *wf-pterm.simps* **by** *fastforce*

with *no-var-lhs* **obtain** *f ts* **where** *lhs*:*lhs* α = *Fun f ts* **by** *fastforce*
from *assms*(*1*) **have** *l1*:*length* (*var-rule* α) = *length As*
using *wf-pterm.cases* **by** *force*
then have *l2*:*length* (*var-poss-list* (*lhs* α)) = *length As*
using *left-lin.length-var-rule*[*OF left-lin-axioms* α] **by** (*simp add*: *length-var-poss-list*)

from *left-lin* **have** *var-rule*:*var-rule* α = *vars-term-list* (*lhs* α)
using α *left-linear-trs-def linear-term-var-vars-term-list* **by** *fastforce*
then show *?thesis* **proof**(*cases p*=[])
case *True*
from *assms*(*3*) **have** β = α **unfolding** *True labeled-source.simps lhs label-term.simps eval-term.simps subt-at.simps* **by** *simp*
then show *?thesis* **unfolding** *True* **by** *simp*
next
case *False*
from *assms*(*3*) **have** *possL*:*p* ∈ *possL* (*Prule* α *As*)
by (*metis assms*(*2*) *get-label-imp-labelposs labeled-source-to-term option.distinct*(*1*) *poss-term-lab-to-term*)
{assume *p* ∈ *fun-poss* (*lhs* α)
then have *get-label* (*labeled-source* (*Prule* α *As*) |- *p*) = *Some* (α, *length p*)
unfolding *labeled-source.simps lhs* **using** *label-term-increase* **by** (*metis add-0*)
with *assms*(*3*) *False* **have** *False* **by** *simp*
}
with *assms*(*2*) **obtain** *p1 p2 x* **where** *p*:*p* = *p1*@*p2* **and** *p1*:*p1* ∈ *poss* (*lhs* α) *lhs* α |- *p1* = *Var x* **and** *p2*:*p2* ∈ *poss* ((⟨*map source As*⟩$_\alpha$) *x*)
unfolding *source.simps* **using** *poss-subst-apply-term*[*of p lhs* α] **by** *metis*
then have *p1* ∈ *var-poss* (*lhs* α) **using** *var-poss-iff* **by** *blast*
with *p1* **obtain** *i* **where** *i*:*i* < *length As vars-term-list* (*lhs* α) !*i* = *x var-poss-list* (*lhs* α) ! *i* = *p1*
using *l2* **by** (*metis in-set-conv-nth length-var-poss-list term.inject*(*1*) *var-poss-list-sound vars-term-list-var-poss-list*)
with *p2 l1* **have** *p2-poss*:*p2* ∈ *poss* (*source* (*As*!*i*))
by (*smt* (*verit*, *del-insts*) α *case-prodD left-lin left-linear-trs-def length-map lhs-subst-var-i linear-term-var-vars-term-list nth-map*)
from *p1* **have** *labeled-source* (*Prule* α *As*) |- *p* = ((⟨*map labeled-source As*⟩$_\alpha$) *x*)|-*p2*
unfolding *labeled-source.simps p* **by** (*smt* (*verit*) *assms*(*2*) *eval-term.simps*(*1*) *label-term-to-term labeled-source.simps*(*3*) *labeled-source-to-term p poss-term-lab-to-term subt-at-subst subterm-poss-conv var-label-term*)
moreover from *var-rule* **have** *map* (⟨*map labeled-source As*⟩$_\alpha$) (*vars-term-list* (*lhs* α)) = *map labeled-source As*
by (*metis apply-lhs-subst-var-rule l1 length-map*)
ultimately have *labeled-source* (*Prule* α *As*) |- *p* = (*labeled-source* (*As*!*i*))|-*p2*
using *i* **by** (*metis map-nth-conv*)
with *assms*(*3*) **have** *get-label* (*labeled-source* (*As* ! *i*) |- *p2*) = *Some* (β, *0*) **by** *force*
with *p2-poss i p* **show** *?thesis* **by** *blast*
qed

qed
end

If the labeled source of a proof term A has the shape $t \cdot \sigma$ where all function symbols in t are unlabeled, then A matches t with some substitution τ.

context *no-var-lhs*
begin
lemma *pterm-source-substitution*:
assumes $A \in$ *wf-pterm R*
  and *source* $A = t \cdot \sigma$ **and** *linear-term t*
  and $\forall\, p \in$ *fun-poss t*. $p \notin$ *possL A*
shows $A =$ (*to-pterm t*) $\cdot$ (*mk-subst Var* (*match-substs* (*to-pterm t*) *A*))
  using *assms* **proof**(*induct A arbitrary*:*t* σ)
  case (*1 x*)
  from *1*(*1*) **obtain** *y* **where** *y*:*t* = *Var y*
    using *subst-apply-eq-Var* **by** (*metis source.simps*(*1*))
  have *match*:*match-substs* (*Var y*) (*Var x*) = [(*y*, *Var x*)]
    unfolding *match-substs-def vars-term-list.simps poss-list.simps* **by** *simp*
  show *?case* **unfolding** *y to-pterm.simps match*
    by *simp*
next
  case (*2 As f*)
  show *?case* **proof**(*cases t*)
    case (*Var x*)
    have *match*:*match-substs* (*Var x*) (*Pfun f As*) = [(*x*, *Pfun f As*)]
      unfolding *match-substs-def vars-term-list.simps poss-list.simps* **by** *simp*
    then show *?thesis* **unfolding** *Var to-pterm.simps match* **by** *simp*
  next
    case (*Fun g ts*)
    from *2*(*2*) **have** *f*:*f* = *g*
      unfolding *Fun* **by** *simp*
    from *2*(*2*) **have** *l*:*length ts* = *length As*
      unfolding *Fun eval-term.simps* **using** *map-eq-imp-length-eq* **by** *fastforce*
    {fix *i* **assume** *i*:*i* < *length As*
      from *2*(*2*) *i* **have** *source* (*As*!*i*) = (*ts*!*i*) $\cdot$ σ
          unfolding *Fun f* **by** (*smt* (*verit*, *best*) *eval-term.simps*(*2*) *l nth-map source.simps*(*2*) *term.sel*(*4*))
      moreover from *2*(*3*) *i l* **have** *lin-tsi*:*linear-term* (*ts*!*i*)
        unfolding *Fun* **by** *simp*
      moreover have ($\forall\, p{\in}$*fun-poss* (*ts*!*i*). $p \notin$ *possL* (*As*!*i*)) **proof**
        fix *p* **assume** *p* ∈ *fun-poss* (*ts*!*i*)
        then have *i*#*p* ∈ *fun-poss* (*Fun f ts*)
          using *i l* **by** *simp*
        with *2*(*4*) **have** *i*#*p* ∉ *possL* (*Pfun f As*)
          unfolding *Fun f* **by** *fastforce*
        then show *p* ∉ *possL* (*As*!*i*)
          using *i* **unfolding** *labeled-source.simps labelposs.simps* **by** *simp*
      qed
       ultimately have *IH*:*As*!*i* = *to-pterm* (*ts*!*i*) $\cdot$ *mk-subst Var* (*match-substs*

(*to-pterm* (*ts*!*i*)) (*As*!*i*))
using *2*(*1*) *i nth-mem* **by** *blast*
have *As*!*i* = *to-pterm* (*ts*!*i*) · *mk-subst Var* (*match-substs* (*to-pterm t*) (*Pfun g As*)) **proof**−
{fix *x* **assume** *x* ∈ *vars-term* (*to-pterm* (*ts*!*i*))
then obtain *j* **where** *j*:*vars-term-list* (*ts*!*i*) !*j* = *x j* < *length* (*vars-term-list* (*ts*!*i*))
by (*metis in-set-conv-nth set-vars-term-list vars-to-pterm*)
then have *j*′:*j* < *length* (*map* ((|-) (*As* ! *i*)) (*var-poss-list* (*to-pterm* (*ts* ! *i*))))
by (*metis length-map length-var-poss-list var-poss-list-to-pterm*)
let *?qj*=*var-poss-list* (*to-pterm* (*ts*!*i*)) !*j*
have *map-j*:(*map* ((|-) (*As* ! *i*)) (*var-poss-list* (*to-pterm* (*ts* ! *i*))))!*j* = (*As*!*i*)|-*?qj*
using *j*′ **by** *simp*
have *distinct* (*vars-term-list* (*ts*!*i*))
using *lin-tsi* **by** (*metis distinct-remdups distinct-rev linear-term-var-vars-term-list o-apply*)
then have *dist1*:*distinct* (*map fst* (*match-substs* (*to-pterm* (*ts*!*i*)) (*As*!*i*)))
unfolding *match-substs-def* **by** (*metis length-map length-var-poss-list map-fst-zip vars-to-pterm*)
have *distinct* (*vars-term-list t*)
by (*metis 2*.*prems*(*2*) *distinct-remdups distinct-rev linear-term-var-vars-term-list o-apply*)
then have *dist2*:*distinct* (*map fst* (*match-substs* (*to-pterm t*) (*Pfun g As*)))
unfolding *match-substs-def* **by** (*metis length-map length-var-poss-list map-fst-zip vars-to-pterm*)
have (*x*, (*As*!*i*)|-*?qj*) ∈ *set* (*match-substs* (*to-pterm* (*ts*!*i*)) (*As*!*i*))
unfolding *match-substs-def* **using** *map-j j j*′ **by** (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-zip min-less-iff-conj nth-zip vars-to-pterm*)
then have *sub1*:*mk-subst Var* (*match-substs* (*to-pterm* (*ts*!*i*)) (*As*!*i*)) *x* = *As*!*i* |- *?qj*
using *dist1 map-of-eq-Some-iff* **unfolding** *mk-subst-def* **by** *simp*
let *?j*′=(*sum-list* (*map* (*length* ∘ *vars-term-list*) (*take i ts*)) + *j*)
have *x2*:*vars-term-list t* ! *?j*′ = *x*
unfolding *Fun vars-term-list.simps* **using** *j*(*1*) **by** (*smt* (*verit*, *best*) *concat-nth i j*(*2*) *l length-map map-map nth-map take-map*)
have *lj*′:*?j*′ < *length* (*vars-term-list* (*to-pterm t*)) **unfolding** *vars-to-pterm* **unfolding** *Fun to-pterm.simps vars-term-list.simps*
using *i j*(*2*) *l concat-nth-length* **by** (*metis List.map.compositionality length-map nth-map take-map*)
then have *j*′*-var-poss*:*?j*′ < *length* (*var-poss-list* (*to-pterm t*))
by (*metis length-var-poss-list*)
then have *lj*′′:*?j*′ < *length* (*map* ((|-) (*Pfun g As*)) (*var-poss-list* (*to-pterm t*)))
by (*metis length-map length-var-poss-list*)
have *var-poss-list* (*to-pterm t*) ! *?j*′ = *i*#*?qj*
proof−
have *l-zip*:*i* < *length* (*zip* [*0*..<*length* (*map to-pterm ts*)] (*map var-poss-list*

(map to-pterm ts)))
by *(simp add: i l)*
have *zip:zip [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)) ! i = (i, var-poss-list (to-pterm (ts ! i)))*
using *nth-zip* **by** *(simp add: i l)*
have *map2:map2 (λi. map ((#) i)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)) ! i ! j = i#?qj*
unfolding *nth-map[OF l-zip] zip* **using** *j'* **by** *auto*
from *l-zip* **have** *i'':i < length (map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)))*
by *simp*
have *j'':j < length (map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)) ! i)*
unfolding *nth-map[OF l-zip] zip* **using** *j(2)* **by** *(metis case-prod-conv length-map length-var-poss-list vars-to-pterm)*
{fix *k* **assume** *k:k < length ts*
then have *zip:zip [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)) ! k = (k, var-poss-list (to-pterm (ts ! k)))*
using *nth-zip* **by** *simp*
then have *map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)) ! k =*
map ((#) k) (var-poss-list (to-pterm (ts ! k)))
using *k* **by** *simp*
then have *length ((map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)))!k) =*
length (vars-term-list (ts!k))
using *length-var-poss-list vars-to-pterm* **by** *(metis length-map)*
with *k* **have** *(map length (map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts))))!k =*
(map (length ∘ vars-term-list) ts) ! k **by** *simp*
}
moreover have *length (map length (map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)))) = length ts*
by *simp*
ultimately have *(map length (map2 (λx. map ((#) x)) [0..<length (map to-pterm ts)] (map var-poss-list (map to-pterm ts)))) =*
(map (length ∘ vars-term-list) ts) **by** *(simp add: map-nth-eq-conv)*
then show *?thesis*
unfolding *Fun to-pterm.simps var-poss-list.simps* **using** *concat-nth[OF i'' j'']* **unfolding** *map2 take-map[symmetric]* **by** *simp*
qed
with *lj''* **have** *(map ((|-) (Pfun g As)) (var-poss-list (to-pterm t)))!?j' = Pfun g As |- (i#?qj)*
by *force*
with *x2* **have** *(x, Pfun g As |-(i#?qj)) ∈ set (match-substs (to-pterm t) (Pfun g As))*
unfolding *match-substs-def* **using** *lj' lj''* **by** *(metis (no-types, lifting) in-set-conv-nth length-zip min-less-iff-conj nth-zip vars-to-pterm)*
then have *sub2:mk-subst Var (match-substs (to-pterm t) (Pfun g As)) x*

```
= Pfun g As |- (i#?qj)
          using dist2 map-of-eq-Some-iff unfolding mk-subst-def by simp
        from sub1 sub2 have mk-subst Var (match-substs (to-pterm (ts!i)) (As!i))
x = mk-subst Var (match-substs (to-pterm t) (Pfun g As)) x
          by simp
       }
       then show ?thesis using IH
         using term-subst-eq by force
     qed
    }
    then show ?thesis
    unfolding Fun f eval-term.simps to-pterm.simps using l by (metis (mono-tags,
lifting) length-map map-nth-eq-conv)
  qed
next
  case (3 α As)
  show ?case proof(cases t)
    case (Var x)
    have match:match-substs (Var x) (Prule α As) = [(x, Prule α As)]
      unfolding match-substs-def vars-term-list.simps poss-list.simps by simp
    then show ?thesis unfolding Var to-pterm.simps match by simp
  next
    case (Fun g ts)
    from 3(1) no-var-lhs obtain f ss where lhsa:lhs α = Fun f ss
      by blast
    have [] ∈ possL (Prule α As)
    unfolding labeled-source.simps lhsa label-term.simps labelposs.simps eval-term.simps
by simp
    with 3(6) have False unfolding Fun by simp
    then show ?thesis by simp
  qed
qed

lemma unlabeled-source-to-pterm:
assumes labeled-source A = s · τ
    and linear-term s and A ∈ wf-pterm R
    and labelposs s = {}
 shows ∃ As. A = to-pterm (term-lab-to-term s) · (mk-subst Var (zip (vars-term-list
s) As)) ∧ length (vars-term-list s) = length As
  using assms proof(induct s arbitrary:A)
  case (Var x)
  let ?As =[A]
  have A = to-pterm (term-lab-to-term (Var x)) · mk-subst Var (zip (vars-term-list
(Var x)) ?As)
   unfolding term-lab-to-term.simps to-pterm.simps vars-term-list.simps zip-Cons-Cons
zip-Nil mk-subst-def by simp
  then show ?case
    by (smt (verit) length-nth-simps(1) list.size(4) vars-term-list.simps(1))
next
```

case (*Fun fl ts*)
from *Fun*(*5*) **obtain** *f* **where** *f*:*fl* = (*f*, *None*)
by (*metis empty-iff empty-pos-in-poss get-label.simps*(*2*) *get-label-imp-labelposs prod.exhaust-sel subt-at.simps*(*1*))
with *Fun*(*2*) **have** ∃ *As*. *A* = *Pfun f As* ∧ *length As* = *length ts* **proof**(*cases A*)
case (*Pfun g As*)
from *Fun*(*2*) **show** *?thesis*
unfolding *Pfun f labeled-source.simps* **using** *map-eq-imp-length-eq* **by** *auto*
next
case (*Prule α As*)
from *Fun*(*4*) *no-var-lhs* **obtain** *g ss* **where** *lhs*:*lhs α* = *Fun g ss*
by (*metis Inl-inject Prule case-prodD is-FunE is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *term.distinct*(*1*) *term.sel*(*2*) *wf-pterm.simps*)
from *Fun*(*2*) **show** *?thesis*
unfolding *Prule f lhs labeled-source.simps* **by** *force*
qed *simp*
then obtain *As* **where** *as*:*A* = *Pfun f As* **and** *l*:*length As* = *length ts*
by *blast*
{fix *i* **assume** *i*:*i* < *length ts*
with *Fun*(*2*) **have** *labeled-source* (*As*!*i*) = (*ts*!*i*) · *τ*
unfolding *as* **by** (*smt* (*verit*, *best*) *eval-term.simps*(*2*) *l labeled-source.simps*(*2*) *nth-map term.inject*(*2*))
moreover from *i Fun*(*3*) **have** *linear-term* (*ts*!*i*)
by *simp*
moreover from *i Fun*(*4*) **have** *As*!*i* ∈ *wf-pterm R*
unfolding *as* **by** (*metis l fun-well-arg nth-mem*)
moreover from *i Fun*(*5*) **have** *labelposs* (*ts*!*i*) = {}
unfolding *f labelposs.simps* **by** *blast*
ultimately have ∃ *As′*. (*As*!*i*) = *to-pterm* (*term-lab-to-term* (*ts*!*i*)) · *mk-subst Var* (*zip* (*vars-term-list* (*ts*!*i*)) *As′*) ∧ *length* (*vars-term-list* (*ts*!*i*)) = *length As′*
using *Fun*(*1*) *i* **by** *force*
}
then obtain *As′* **where** *l′′*:*length As′* = *length ts*
and *IH*:(∀ *i* < *length ts*. (*As*!*i*) = *to-pterm* (*term-lab-to-term* (*ts*!*i*)) · *mk-subst Var* (*zip* (*vars-term-list* (*ts*!*i*)) (*As′*!*i*)) ∧ *length* (*vars-term-list* (*ts*!*i*)) = *length* (*As′*!*i*))
using *Ex-list-of-length-P*[**where** *P*=λ*As′ i*. *As* ! *i* = *to-pterm* (*term-lab-to-term* (*ts* ! *i*)) · *mk-subst Var* (*zip* (*vars-term-list* (*ts* ! *i*)) *As′*) ∧ *length* (*vars-term-list* (*ts*!*i*)) = *length As′*] *l* **by** *blast*
then have *l′*:*length As′* = *length* (*map to-pterm* (*map term-lab-to-term ts*))
by *simp*
have *vars-list*:*map vars-term-list* (*map to-pterm* (*map term-lab-to-term ts*)) = *map vars-term-list ts*
by (*smt* (*verit*, *best*) *length-map map-nth-eq-conv vars-term-list-term-lab-to-term vars-to-pterm*)
have *map vars-term* (*map to-pterm* (*map term-lab-to-term ts*)) = *map vars-term ts*
using *vars-term-list-term-lab-to-term* **by** (*smt* (*verit*, *ccfv-threshold*) *length-map map-nth-eq-conv set-vars-term-list vars-to-pterm*)

then have *part*:*is-partition* (*map vars-term* (*map to-pterm* (*map term-lab-to-term ts*)))
using *Fun*(*3*) **by** (*metis linear-term.simps*(*2*))
have ∗:∀ *i* < *length ts*. *to-pterm* (*term-lab-to-term* (*ts*!*i*)) · *mk-subst Var* (*concat* (*map2 zip* (*map vars-term-list* (*map to-pterm* (*map term-lab-to-term ts*))) *As*′)) = *As*!*i*
using *mk-subst-partition-special*[*OF l*′ *part*] **unfolding** *length-map* **using** *nth-map IH*
by (*smt* (*verit*, *best*) *length-map vars-term-list-term-lab-to-term vars-to-pterm*)
from *IH* **have** ∀ *i* < *length ts*. *length* (*vars-term-list* (*to-pterm* (*term-lab-to-term* (*ts*! *i*)))) = *length* (*As*′ ! *i*)
by (*metis vars-term-list-term-lab-to-term vars-to-pterm*)
then have *ls*:∀ *i* < *length ts*. *length* (*map vars-term-list* (*map to-pterm* (*map term-lab-to-term ts*)) ! *i*) = *length* (*As*′ ! *i*)
using *nth-map* **by** *simp*
then have *cc*:*concat* (*map2 zip* (*map vars-term-list* (*map to-pterm* (*map term-lab-to-term ts*))) *As*′) = *zip* (*concat* (*map vars-term-list ts*)) (*concat As*′)
unfolding *vars-list* **using** *concat-map2-zip* **by** (*metis l*′ *length-map*)
have *A* = *to-pterm* (*term-lab-to-term* (*Fun fl ts*)) · *mk-subst Var* (*zip* (*vars-term-list* (*Fun fl ts*)) (*concat As*′))
unfolding *f term-lab-to-term.simps to-pterm.simps fst-conv eval-term.simps as vars-term-list.simps cc*[*symmetric*] **using** ∗ **by** (*simp add*: *l list-eq-iff-nth-eq*)
moreover have *length* (*vars-term-list* (*Fun fl ts*)) = *length* (*concat As*′)
unfolding *vars-term-list.simps*
using *l*′′ *ls* **by** (*metis eq-length-concat-nth length-map vars-list*)
ultimately show *?case* **by** *auto*
qed
end

lemma *labels-intersect-label-term*:
assumes *term-lab-to-term A* = *t* · (*term-lab-to-term* ∘ *σ*)
and *linear-term t*
and *labelposs A* ∩ *labelposs* ((*label-term α n t*) · *σ*) = {}
shows ∃ *As*. *A* = *term-to-term-lab t* · (*mk-subst Var* (*zip* (*vars-term-list t*) *As*)) ∧ *length As* = *length* (*vars-term-list t*)
using *assms* **proof**(*induct t arbitrary*:*A n*)
case (*Var x*)
have *A* = *mk-subst Var* (*zip* [*x*] [*A*]) *x*
unfolding *mk-subst-def* **by** *simp*
then show *?case* **unfolding** *term-to-term-lab.simps eval-term.simps vars-term-list.simps* **by** *fastforce*
next
case (*Fun f ts*)
from *Fun*(*2*) **obtain** *lab ss* **where** *a*:*A* = *Fun* (*f*, *lab*) *ss*
using *term-lab-to-term.simps* **by** (*smt* (*verit*, *ccfv-threshold*) *eroot.cases fst-conv old.prod.exhaust eval-term.simps*(*2*) *term.distinct*(*1*) *term.sel*(*2*))
from *Fun*(*4*) **have** *lab*:*lab* = *None*
unfolding *a* **using** *insertCI* **by** *auto*
from *Fun*(*2*) **have** *l*:*length ts* = *length ss*

unfolding *a* **by** (*metis length-map eval-term.simps*(*2*) *term.sel*(*4*) *term-lab-to-term.simps*(*2*))
{fix *i* **assume** *i*:*i* < *length ts*
with *Fun*(*2*) **have** *term-lab-to-term* (*ss*!*i*) = *ts*!*i* · (*term-lab-to-term* ∘ *σ*)
unfolding *a term-lab-to-term.simps eval-term.simps fst-conv* **by** (*metis l nth-map term.inject*(*2*))
moreover from *i Fun*(*3*) **have** *linear-term* (*ts*!*i*)
by *simp*
moreover have *labelposs* (*ss*!*i*) ∩ *labelposs* (*label-term α* (*n*+*1*) (*ts*!*i*) · *σ*) = {}
proof−
{fix *q* **assume** *q1*:*q* ∈ *labelposs* (*ss*!*i*) **and** *q2*:*q* ∈ *labelposs* (*label-term α* (*n*+*1*) (*ts*!*i*) · *σ*)
from *q1 i l* **have** *i*#*q* ∈ *labelposs A*
unfolding *a lab label-term.simps labelposs.simps* **by** *simp*
moreover from *q2 i* **have** *i*#*q* ∈ *labelposs* ((*label-term α n* (*Fun f ts*)) · *σ*)
unfolding *label-term.simps eval-term.simps labelposs.simps length-map* **by** *simp*
ultimately have *False*
using *Fun*(*4*) **by** *blast*
}
then show *?thesis*
by *blast*
qed
ultimately have ∃ *As*. *ss*!*i* = *term-to-term-lab* (*ts*!*i*) · (*mk-subst Var* (*zip* (*vars-term-list* (*ts*!*i*)) *As*)) ∧ *length As* = *length* (*vars-term-list* (*ts*!*i*))
using *Fun*(*1*) *i nth-mem* **by** *blast*
}
then obtain *Ass* **where** *l*′:*length Ass* = *length ts*
and *IH*:(∀ *i* < *length ts*. (*ss*!*i*) = (*term-to-term-lab* (*ts*!*i*)) · *mk-subst Var* (*zip* (*vars-term-list* (*ts*!*i*)) (*Ass*!*i*)) ∧ *length* (*Ass*!*i*) = *length* (*vars-term-list* (*ts*!*i*)))
using *Ex-list-of-length-P*[**where** *P*=λ*Ass i*. *ss* ! *i* = (*term-to-term-lab* (*ts* ! *i*)) · *mk-subst Var* (*zip* (*vars-term-list* (*ts* ! *i*)) *Ass*) ∧ *length Ass* = *length* (*vars-term-list* (*ts*!*i*))] *l* **by** *blast*
let *?As*=*concat Ass*
from *l*′ **have** *l*′′:*length Ass* = *length* (*map term-to-term-lab ts*)
by *simp*
have *vars-list*:*map vars-term-list* (*map term-to-term-lab ts*) = *map vars-term-list ts*
using *vars-term-list-term-to-term-lab* **by** *auto*
have *map vars-term* (*map term-to-term-lab ts*) = *map vars-term ts*
using *vars-term-list-term-to-term-lab* **by** (*smt* (*verit*, *ccfv-threshold*) *length-map map-nth-eq-conv set-vars-term-list vars-to-pterm*)
then have *part*:*is-partition* (*map vars-term* (*map term-to-term-lab ts*))
using *Fun*(*3*) **by** (*metis linear-term.simps*(*2*))
have ∗:∀ *i* < *length ts*. (*term-to-term-lab* (*ts*!*i*)) · *mk-subst Var* (*concat* (*map2 zip* (*map vars-term-list* (*map term-to-term-lab ts*)) *Ass*)) = *ss*!*i*
using *mk-subst-partition-special*[*OF l*′′ *part*] **unfolding** *length-map* **using** *nth-map IH*
by (*smt* (*verit*, *best*) *length-map vars-term-list-term-to-term-lab vars-to-pterm*)

```
  from IH have ∀ i < length ts. length (vars-term-list (term-to-term-lab (ts! i)))
= length (Ass ! i)
    by (metis vars-term-list-term-to-term-lab)
  then have ls:∀ i < length ts. length (map vars-term-list (map term-to-term-lab
ts) ! i) = length (Ass ! i)
    using nth-map by simp
  then have cc:concat (map2 zip (map vars-term-list (map term-to-term-lab ts))
Ass) = zip (concat (map vars-term-list ts)) (concat Ass)
    unfolding vars-list using concat-map2-zip by (metis l' length-map)
  have A = term-to-term-lab (Fun f ts) · mk-subst Var (zip (vars-term-list (Fun f
ts)) ?As)
     unfolding term-to-term-lab.simps eval-term.simps vars-term-list.simps a lab
cc[symmetric] using ∗ by (simp add: l list-eq-iff-nth-eq)
  moreover from IH l' have l'':length ?As = length (vars-term-list (Fun f ts))
    unfolding vars-term-list.simps by (simp add: eq-length-concat-nth)
  ultimately show ?case
    by blast
qed

lemma labeled-wf-pterm-rule-in-TRS:
  assumes A ∈ wf-pterm R and p ∈ poss (labeled-source A)
    and get-label (labeled-source A |- p) = Some (α, n)
  shows to-rule α ∈ R
  using assms proof(induct A arbitrary: p n)
  case (2 ts f)
  from 2(2,3) obtain i p' where p:p = i#p' i < length ts p' ∈ poss (labeled-source
(ts!i)) get-label (labeled-source (ts!i) |- p') = Some (α, n)
    unfolding labeled-source.simps get-label.simps by auto
  with 2(1) show ?case
    using nth-mem by blast
next
  case (3 β As)
  from 3(4) consider p ∈ fun-poss (labeled-lhs β) | (∃ p1 p2 x. p = p1@p2
                                  ∧ p1 ∈ poss (labeled-lhs β) ∧ (labeled-lhs β)|-p1 =
Var x
                                  ∧ p2 ∈ poss ((⟨map labeled-source As⟩β) x)
                                      ∧ (labeled-source (Prule β As))|-p = ((⟨map
labeled-source As⟩β) x)|-p2)
   unfolding labeled-source.simps by (meson poss-is-Fun-fun-poss poss-subst-choice)
  then show ?case proof(cases)
    case 1
    then have p ∈ fun-poss (lhs β)
      by (simp add: fun-poss-label-term)
    then have get-label ((labeled-source (Prule β As))|-p) = Some (β, length p)
      unfolding labeled-source.simps by (simp add: label-term-increase)
    with 3(1,5) show ?thesis by auto
  next
    case 2
    then obtain p1 p2 x where p1p2:p = p1 @ p2 and x:p1 ∈ poss (labeled-lhs
```

```
β) ∧ labeled-lhs β |- p1 = Var x
      and p2:p2 ∈ poss ((⟨map labeled-source As⟩β) x)
      and lab:labeled-source (Prule β As) |- p = (⟨map labeled-source As⟩β) x |- p2
     by blast
   from x have x ∈ vars-term (lhs β)
    by (metis subt-at-imp-supteq subteq-Var-imp-in-vars-term vars-term-labeled-lhs)
   with x obtain i where i:i < length (var-rule β) ∧ (var-rule β)!i = x
    by (metis in-set-conv-nth set-vars-term-list vars-term-list-vars-distinct)
   with 3(2) have ∗:(⟨map labeled-source As⟩β) x = labeled-source (As!i)
    by (metis (no-types, lifting) length-map lhs-subst-var-i nth-map)
   with 3(5) lab have get-label ((labeled-source (As!i))|-p2) = Some (α, n)
    by simp
   with 3(3) p2 i 3(2) ∗ show ?thesis by force
  qed
qed simp

context no-var-lhs
begin
lemma unlabeled-above-p:
  assumes A ∈ wf-pterm R
    and p ∈ poss (source A)
    and ∀ r. r <p p ⟶ r ∉ possL A
  shows p ∈ poss A ∧ labeled-source A|-p = labeled-source (A|-p)
  using assms proof(induct p arbitrary: A)
  case (Cons i p)
  from Cons(3) obtain f ts where f:source A = Fun f ts and i:i < length ts and
p:p ∈ poss (ts!i)
    using args-poss by blast
  from Cons(4) have [] ∉ possL A
    by simp
  then have no-lab:get-label (labeled-source A) = None
    by (metis empty-pos-in-poss get-label-imp-labelposs subt-at.simps(1))
  from Cons(3) obtain f' As where a:A = Fun f' As
    by (metis Cons-poss-Var eroot.cases source.simps(1))
  then have f':f' = Inr f proof(cases A)
    case (Pfun g Bs)
    then show ?thesis using f Pfun a by simp
  next
    case (Prule α Bs)
    with Cons(2) obtain g ss where g:lhs α = Fun g ss
     using no-var-lhs by (metis Inl-inject case-prodD is-Prule.simps(1) is-Prule.simps(3)
term.collapse(2) term.distinct(1) term.sel(2) wf-pterm.simps)
    then show ?thesis using no-lab unfolding Prule by simp
  qed simp
  from i a have i':i < length As
    using f f' by force
  from Cons(3) have p ∈ poss (source (As!i))
    unfolding a f' by auto
  moreover
```

```
  {fix r assume r ∈ poss (source (As!i)) and le:r <p p
    then have i#r ∈ poss (labeled-source A)
      unfolding a f' using i' by simp
    moreover from le have i#r <p i#p
      by simp
    ultimately have i#r ∉ possL A
      using Cons(4) by blast
    then have r ∉ possL (As!i)
      unfolding a f' labeled-source.simps using i' by force
  }
  ultimately have p ∈ poss (As!i) ∧ labeled-source (As!i) |- p = labeled-source
((As!i) |- p)
   using Cons(1,2) i' unfolding a f' by (meson fun-well-arg nth-mem possL-subset-poss-source
subsetD)
  with i' a f' show ?case
    by simp
qed simp
end

lemma (in single-redex) labeled-source-at-pq:labeled-source (A|-q) = (labeled-source
A)|-p
  using a pq q p a-well proof(induct q arbitrary:p A)
  case Nil
  then have p = []
    by (simp add: subt-at-ctxt-of-pos-term subt-at-id-imp-eps)
  then show ?case
    by simp
next
  case (Cons i q)
  from Cons(4) obtain fs Bs where a:A = Fun fs Bs and i:i < length Bs and
q:q ∈ poss (Bs!i)
    using args-poss by blast
  let ?As = map (λj. (Bs!i) |- (q @ [j])) [0..<length (var-rule α)]
  have (map (λia. A |- ((i # q) @ [ia])) [0..<length (var-rule α)]) = ?As
    unfolding a by simp
  with a i q Cons(2,4) have bsi:Bs!i = (ctxt-of-pos-term q (Bs!i))⟨Prule α ?As⟩
    by (metis ctxt-supt-id subt-at.simps(2) subt-at-ctxt-of-pos-term)
  from Cons(6) have bi-well:Bs ! i ∈ wf-pterm R
    unfolding a by (meson fun-well-arg i nth-mem)
  show ?case proof(cases fs)
    case (Inl β)
    from Cons(6) have lin:linear-term (lhs β)
     unfolding a Inl using left-lin left-linear-trs-def term.inject(2) wf-pterm.cases
by fastforce
    from Cons(6) have is-Fun:is-Fun (lhs β)
      unfolding a Inl using no-var-lhs using wf-pterm.cases by auto
    from Cons(6) have l-bs:length Bs = length (var-rule β)
      unfolding a Inl using wf-pterm.cases by auto
    obtain p1 p2 where p:p = p1@p2 and p1:p1 = var-poss-list (lhs β) ! i and
```

p2:*p2* ∈ *poss* (*source* (*Bs*!*i*))
using *ctxt-rule-obtain-pos Cons*(*4*,*5*,*3*) *lin l-bs* **unfolding** *a Inl* **by** *metis*
have *ctxt*:*ctxt-of-pos-term p2* (*source* (*Bs* ! *i*)) = *source-ctxt* (*ctxt-of-pos-term q* (*Bs* ! *i*))
proof−
from *p1* **have** *p1-pos*:*p1* ∈ *poss* (*lhs* β)
using *i l-bs lin* **by** (*metis length-var-poss-list linear-term-var-vars-term-list nth-mem var-poss-imp-poss var-poss-list-sound*)
from *p1-pos* **have** *p1*′:*p1* ∈ *poss* (*lhs* β · ⟨*map source Bs*⟩β)
by *simp*
from *p1* **have** *p1*′′:*var-poss-list* (*lhs* β) ! *length* (*take i Bs*) = *p1*
using *i* **by** *force*
have ∗:*lhs* β · ⟨*map source Bs*⟩β |- *p1* = *source* (*Bs*!*i*)
unfolding *p1* **using** *l-bs i*
by (*smt* (*verit*) *length-map lhs-subst-var-i lin linear-term-var-vars-term-list nth-map p1 p1-pos eval-term.simps*(*1*) *subt-at-subst vars-term-list-var-poss-list*)
from *Cons*(*3*) **show** *?thesis*
unfolding *a Inl p source.simps ctxt-of-pos-term.simps source-ctxt.simps Let-def ctxt-of-pos-term-append*[*OF p1*′] ∗ *p1*′′
using *ctxt-comp-equals*[*OF p1*′] *p1-pos* **using** *poss-imp-subst-poss* **by** *blast*
qed
from *Cons*(*1*)[*OF bsi ctxt q p2 bi-well*] **have** *IH*: *labeled-source* (*Bs* ! *i* |- *q*) = *labeled-source* (*Bs* ! *i*) |- *p2*
by *presburger*
from *p1* **have** *p1* = *var-poss-list* (*labeled-lhs* β) ! *i*
by (*simp add*: *var-poss-list-labeled-lhs*)
moreover then have (*labeled-lhs* β)|-*p1* = *Var* (*vars-term-list* (*lhs* β)!*i*)
by (*metis i l-bs lin linear-term-var-vars-term-list vars-term-list-labeled-lhs vars-term-list-var-poss-list*)
ultimately show *?thesis*
unfolding *a Inl* **using** *i IH* **unfolding** *subt-at.simps p labeled-source.simps*
by (*smt* (*verit*, *ccfv-threshold*) *apply-lhs-subst-var-rule filter-cong l-bs length-map length-var-poss-list lin linear-term-var-vars-term-list map-nth-conv nth-mem poss-imp-subst-poss eval-term.simps*(*1*) *subt-at-append subt-at-subst var-poss-imp-poss var-poss-list-sound vars-term-list-labeled-lhs*)
next
case (*Inr f*)
from *Cons*(*3*,*5*) **obtain** *p*′ **where** *p*:*p* = *i*#*p*′ **and** *p*′:*p*′ ∈ *poss* (*source* (*Bs*!*i*))
by (*metis Cons.prems*(*3*) *Inr a source-poss*)
from *Cons*(*3*) **have** *ctxt*:*ctxt-of-pos-term p*′ (*source* (*Bs* ! *i*)) = *source-ctxt* (*ctxt-of-pos-term q* (*Bs* ! *i*))
unfolding *a Inr p* **by** (*simp add*: *i*)
from *Cons*(*1*)[*OF bsi ctxt q p*′ *bi-well*] **have** *IH*:*labeled-source* (*Bs* ! *i* |- *q*) = *labeled-source* (*Bs* ! *i*) |- *p*′
by *presburger*
then show *?thesis*
unfolding *a Inr p* **using** *i* **by** *simp*
qed
qed

context *left-lin*
begin
lemma *single-redex-label*:
assumes Δ = *ll-single-redex s p* α *p* $\in$ *poss s q* $\in$ *poss* (*source* Δ) *to-rule* $\alpha \in R$
and *get-label* (*labeled-source* Δ |-*q*) = *Some* (β, *n*)
shows $\alpha = \beta \wedge (\exists\, q'.\ q = p@q' \wedge$ *length* $q' = n \wedge q' \in$ *fun-poss* (*lhs* α))
proof−
from *assms* **have** *wf*:$\Delta \in$ *wf-pterm R*
using *single-redex-wf-pterm left-lin left-linear-trs-def* **by** *fastforce*
from *assms* **have** *q* $\in$ *possL* Δ
using *get-label-imp-labelposs* **by** *force*
then obtain *q′* **where** *q*:*q* = *p*@*q′* **and** *q′*:*q′* $\in$ *fun-poss* (*lhs* α)
unfolding *assms*(*1*) **using** *single-redex-possL*[*OF assms*(*4*,*2*)] **by** *auto*
from *assms* **have** *labeled-source* Δ = (*ctxt-of-pos-term p* (*labeled-source* (*to-pterm s*)))⟨*labeled-source* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α)))))⟩
using *label-source-ctxt* **by** (*simp add*: *ll-single-redex-def p-in-poss-to-pterm source-ctxt-to-pterm*)
then have *labeled-source* Δ |-*q* = *labeled-source* (*Prule* α (*map* (*to-pterm* ∘ (λ*pi*. *s*|-(*p*@*pi*))) (*var-poss-list* (*lhs* α)))) |-*q′*
unfolding *q* **using** *assms*(*2*) **by** (*metis hole-pos-ctxt-of-pos-term hole-pos-poss labeled-source-to-term poss-term-lab-to-term replace-at-subt-at source-to-pterm subt-at-append*)

then have *get-label* (*labeled-source* Δ |-*q*) = *get-label* (*labeled-lhs* α|-*q′*)
using *get-label-at-fun-poss-subst q′* **by** *force*
also have ... = *Some* (α, *size q′*)
using *get-label-label-term q′* **by** *fastforce*
finally show *?thesis* **using** *assms q q′* **by** *force*
qed
end

5.2 Measuring Overlap

abbreviation *measure-ov* :: ('*f*, '*v*) *pterm* ⇒ ('*f*, '*v*) *pterm* ⇒ *nat*
where *measure-ov A B* ≡ *card* ((*possL A*) ∩ (*possL B*))

lemma *finite-labelposs*: *finite* (*labelposs A*)
by (*meson finite-fun-poss labelposs-subs-fun-poss rev-finite-subset*)

lemma *finite-possL*: *finite* (*possL A*)
by (*simp add*: *finite-labelposs*)

lemma *measure-ov-symm*: *measure-ov A B* = *measure-ov B A*
by (*simp add*: *Int-commute*)

lemma *measure-lhs-subst*:
assumes *l*:*length As* = *length Bs*
shows *card* ((*labelposs* ((*label-term* α *j t*) · ⟨*map labeled-source As*⟩$_\alpha$)) ∩

```
      (labelposs (labeled-source (to-pterm t) · ⟨map labeled-source Bs⟩α)))
    = (∑ x←vars-term-list t. measure-ov ((⟨As⟩α) x) ((⟨Bs⟩α) x))
  using assms proof(induct t arbitrary:j)
case (Var x)
  show ?case proof(cases ∃ i < length As. i < length (var-rule α) ∧ x = (var-rule
α)!i)
    case True
    then obtain i where i:x = (var-rule α)!i and il:i < length As  and il2:i <
length (var-rule α) by auto
    then have a:(⟨map labeled-source As⟩α) x = labeled-source (As!i)
      using lhs-subst-var-i by (metis (no-types, lifting) length-map nth-map)
    from i il il2 l have b:(⟨map labeled-source Bs⟩α) x = labeled-source (Bs!i)
      using lhs-subst-var-i  by (metis (no-types, lifting) length-map nth-map)
    from i show ?thesis unfolding vars-term-list.simps sum-list-elem
    unfolding to-pterm.simps label-term.simps labeled-source.simps eval-term.simps
      unfolding a b using lhs-subst-var-i l il il2 by metis
  next
    case False
    then have a:(⟨map labeled-source As⟩α) x = Var x
      using lhs-subst-not-var-i by (metis length-map)
    from False l have b:(⟨map labeled-source Bs⟩α) x = Var x
      using lhs-subst-not-var-i by (metis length-map)
    from False l have possL ((⟨As⟩α) x) ∩ possL ((⟨Bs⟩α) x) = {}
      unfolding term.set(3) using lhs-subst-not-var-i
      by (metis inf.idem labeled-source.simps(1) labelposs.simps(1))
   then show ?thesis unfolding label-term.simps to-pterm.simps labeled-source.simps
eval-term.simps a b
      by auto
  qed
next
  case (Fun f ts)
  let ?as=(map (λt. t · ⟨map labeled-source As⟩α) (map (label-term α (j + 1)) ts))
  let ?bs=(map (λt. t · ⟨map labeled-source Bs⟩α) (map labeled-source (map to-pterm
ts)))
  let ?f=(λi. ({i # p |p. p ∈ labelposs (?as ! i)} ∩ {i # p |p. p ∈ labelposs (?bs !
i)}))
   have {[]} ∩ (⋃i<length ts. {i # p |p. p ∈ labelposs (map (λt. t · ⟨map la-
beled-source Bs⟩α) (map labeled-source (map to-pterm ts)) ! i)}) = {}
    by blast
  then have ∗:labelposs (label-term α j (Fun f ts) · ⟨map labeled-source As⟩α) ∩
labelposs (labeled-source (to-pterm (Fun f ts)) · ⟨map labeled-source Bs⟩α)
      = (⋃i<length ts. (?f i))
   unfolding label-term.simps to-pterm.simps labeled-source.simps eval-term.simps
labelposs.simps by auto
  have is-partition (map ?f [0..<length ts]) proof−
    {fix i j assume j:j < length ts and i:i < j
      have ?f i ∩ ?f j = {} unfolding Int-def using i
        by fastforce
    }
```

```
  then show ?thesis unfolding is-partition-def by auto
 qed
 moreover have ∀ i<length ts. finite (?f i) by (simp add: finite-labelposs)
 ultimately have **:card (⋃ i<length ts. (?f i)) = (∑ i<length ts. card (?f i))
  unfolding * using card-Union-Sum by blast
 {fix i assume i:i < length ts
  have ?f i = {i # p |p. p ∈ labelposs (?as ! i) ∩ labelposs (?bs ! i)}
   unfolding Int-def by blast
  then have card (?f i) = card (labelposs (?as ! i) ∩ labelposs (?bs ! i))
   unfolding Setcompr-eq-image using card-image by (metis (no-types, lifting)
inj-on-Cons1)
  with Fun i have card (?f i) = (∑ x←vars-term-list (ts!i). measure-ov ((⟨As⟩α)
x) ((⟨Bs⟩α) x))
   by simp
 }
 then show ?case unfolding vars-term-list.simps * **
  by (simp add: sum-sum-concat)
qed

lemma measure-lhs-args-zero:
 assumes l:length As = length Bs
    and empty:∀ i < length As. measure-ov (As!i) (Bs!i) = 0
  shows measure-ov (Prule α As) ((to-pterm (lhs α)) · ⟨Bs⟩α) = 0
proof−
 let ?xs=vars-term-list (lhs α)
 have sum:measure-ov (Prule α As) ((to-pterm (lhs α)) · ⟨Bs⟩α)
         = (∑ x←vars-term-list (lhs α). measure-ov ((⟨As⟩α) x) ((⟨Bs⟩α) x))
  using labeled-source-apply-subst measure-lhs-subst[OF l]
  by (metis (mono-tags, lifting) fun-mk-subst labeled-source.simps(1) labeled-source.simps(3)
to-pterm-wf-pterm)
 {fix i assume i:i < length ?xs
  have measure-ov ((⟨As⟩α) (?xs ! i)) ((⟨Bs⟩α) (?xs ! i)) = 0
  proof(cases (∃ j<length As. j < length (var-rule α) ∧ (?xs!i) = var-rule α ! j))
   case True
   then obtain j where j:j < length As j < length (var-rule α) and ij:?xs!i =
(var-rule α)!j
    by blast
   then show ?thesis
    unfolding ij using empty by (metis j l lhs-subst-var-i)
  next
   case False
   then have (⟨As⟩α) (?xs!i) = Var (?xs!i)
    using lhs-subst-not-var-i by metis
   moreover have (⟨Bs⟩α) (?xs!i) = Var (?xs!i)
    using l False lhs-subst-not-var-i by metis
   ultimately show ?thesis by simp
  qed}
 then show ?thesis
  using sum by (simp add: sum-list-zero)
```

qed

lemma *measure-zero-subt-at*:
assumes *term-lab-to-term A = term-lab-to-term B*
and *labelposs A ∩ labelposs B = {}*
and *p ∈ poss A*
shows *labelposs (A|-p) ∩ labelposs (B|-p) = {}*
using *assms* **proof**(*induct p arbitrary*: *A B*)
case (*Cons i p*)
from *Cons*(*4*) **obtain** *f a ts* **where** *a*:*A = Fun* (*f*, *a*) *ts* **and** *i*:*i < length ts* **and** *p*:*p ∈ poss* (*ts*!*i*)
using *args-poss* **by** (*metis old.prod.exhaust*)
with *Cons*(*2*) **obtain** *b ss* **where** *b*:*B = Fun* (*f*, *b*) *ss*
by (*metis* (*no-types*, *opaque-lifting*) *Cons.prems*(*3*) *Term.term.simps*(*2*) *args-poss old.prod.exhaust poss-term-lab-to-term prod.sel*(*1*) *term-lab-to-term.simps*(*2*))
have *ts*:(⋃ *i*<*length ts*. {*i* # *p* | *p*. *p ∈ labelposs* (*ts* ! *i*)}) ⊆ *labelposs A* **unfolding** *a* **by**(*cases a*) *auto*
have *ss*:(⋃ *i*<*length ss*. {*i* # *p* | *p*. *p ∈ labelposs* (*ss* ! *i*)}) ⊆ *labelposs B* **unfolding** *b* **by**(*cases b*) *auto*
from *ss ts b i Cons*(*2*,*3*,*4*) **have** *labelposs* (*ts*!*i*) ∩ *labelposs* (*ss*!*i*) = {} **by** *auto*
with *Cons*(*1*,*2*) *p i* **show** *?case*
unfolding *a b* **by** (*simp add*: *map-eq-conv'*)
qed *simp*

lemma *empty-step-imp-measure-zero*:
assumes *is-empty-step A*
shows *measure-ov A B = 0*
by (*metis assms card-eq-0-iff inf-bot-left labeled-source-simple-pterm source-empty-step*)

lemma *measure-ov-to-pterm*:
shows *measure-ov A* (*to-pterm t*) = *0*
by (*simp add*: *labeled-source-simple-pterm*)

lemma *measure-zero-imp-orthogonal*:
assumes *R*:*left-lin-no-var-lhs R* **and** *S*:*left-lin-no-var-lhs S*
and *co-initial A B A ∈ wf-pterm R B ∈ wf-pterm S*
and *measure-ov A B = 0*
shows *A* $\perp_p$ *B*
using *assms*(*3*−) **proof**(*induct A arbitrary*:*B rule*:*subterm-induct*)
case (*subterm A*)
then show *?case* **proof**(*cases A*)
case (*Var x*)
with *subterm* **show** *?thesis* **proof**(*cases B*)
case (*Prule α Bs*)
from *subterm*(*2*) *Var* **obtain** *y* **where** *y*:*lhs α = Var y*
unfolding *Prule* **by** (*metis source.simps*(*1*) *source.simps*(*3*) *subst-apply-eq-Var*)
from *subterm*(*4*) *Prule S* **have** *is-Fun* (*lhs α*)
unfolding *left-lin-no-var-lhs-def no-var-lhs-def*

```
      by (metis Inl-inject case-prodD is-FunI is-Prule.simps(1) is-Prule.simps(3)
is-VarI term.inject(2) wf-pterm.simps)
    with y show ?thesis by simp
  qed (simp-all add: orthogonal.intros(1))
 next
  case (Pfun f As)
  note A=this
  with subterm show ?thesis proof(cases B)
   case (Pfun g Bs)
   from subterm(2) have f:f = g
    unfolding Pfun A by simp
   from subterm(2) have l:length As = length Bs
    unfolding A Pfun using map-eq-imp-length-eq by auto
   {fix i assume i:i < length As
    then have As!i ◁ A
     unfolding A by simp
    moreover from i subterm(2) l have co-initial (As!i) (Bs!i)
     by (metis (mono-tags, lifting) A Pfun nth-map source.simps(2) term.inject(2))
    moreover from i subterm(3) have As!i ∈ wf-pterm R
     using A by auto
    moreover from i subterm(4) l have Bs!i ∈ wf-pterm S
     using Pfun by auto
    moreover have measure-ov (As!i) (Bs!i) = 0 proof−
     {fix p assume a:p ∈ possL (As!i) and b:p ∈ possL (Bs!i)
      with i have i#p ∈ possL A
       unfolding A labeled-source.simps labelposs.simps by simp
      moreover from b i l have i#p ∈ possL B
       unfolding Pfun labeled-source.simps labelposs.simps by simp
      ultimately have False using subterm(4)
     by (metis card-gt-0-iff disjoint-iff finite-Int finite-possL less-numeral-extra(3)
subterm.prems(4))
     }
     then show ?thesis
      by (metis card.empty disjoint-iff)
    qed
    ultimately have As!i ⊥p Bs!i
     using subterm(1) by blast
   }
   then show ?thesis
    unfolding A Pfun f using l by auto
  next
   case (Prule β Bs)
   from subterm(4) S have lin:linear-term (lhs β)
     unfolding Prule left-lin-no-var-lhs-def left-lin-def left-linear-trs-def using
wf-pterm.cases by fastforce
   have isfun:is-Fun (lhs β)
   using subterm(4) S no-var-lhs.lhs-is-Fun unfolding Prule left-lin-no-var-lhs-def
by blast
   have (lhs β) · (term-lab-to-term ∘ (⟨map labeled-source Bs⟩β)) = lhs β · ⟨map
```

source Bs$\rangle_\beta$
by (*metis label-term-to-term labeled-source.simps*(3) *labeled-source-to-term source.simps*(3) *term-lab-to-term-subst*)
with *subterm*(2) **have** *co-init*:*term-lab-to-term* (*labeled-source A*) = *lhs* β · (*term-lab-to-term* ∘ $\langle$*map labeled-source Bs*$\rangle_\beta$)
unfolding *Prule* **by** *simp*
from *subterm*(5) **have** *possL A* ∩ *possL B* = {}
by (*simp add*: *finite-possL*)
then obtain τ **where** *labeled-source A* = *term-to-term-lab* (*lhs* β) · τ
unfolding *labeled-source.simps*(3) *Prule* **using** *labels-intersect-label-term*[*OF co-init lin*] **by** *blast*
moreover have *labelposs* (*term-to-term-lab* (*lhs* β)) = {}
using *labelposs-term-to-term-lab* **by** *blast*
moreover from *lin* **have** *linear-term* (*term-to-term-lab* (*lhs* β))
using *linear-term-to-term-lab* **by** *blast*
moreover have *term-lab-to-term* (*term-to-term-lab* (*lhs* β)) = *lhs* β
by *simp*
ultimately obtain σ **where** *sigma*:*A* = *to-pterm* (*lhs* β) · σ
using *no-var-lhs.unlabeled-source-to-pterm subterm*(3) *R* **unfolding** *left-lin-no-var-lhs-def* **by** *metis*
let *?As*=*map* σ (*var-rule* β)
from *sigma* **have** *a*:*A* = (*to-pterm* (*lhs* β)) · $\langle$*?As*$\rangle_\beta$
by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule comp-apply length-map map-eq-conv set-remdups set-rev set-vars-term-list term-subst-eq vars-to-pterm*)
{fix *i* **assume** *i*:*i* < *length* (*var-rule* β)
let *?xi*=*var-rule* β!*i*
from *i* **obtain** *i*′ **where** *i*′:*i*′ < *length* (*vars-term-list* (*lhs* β)) *?xi* = *vars-term-list* (*lhs* β)!*i*′
by (*metis comp-apply in-set-conv-nth set-remdups set-rev*)
have *l*:*length Bs* = *length* (*var-rule* β)
using *subterm*(4) **unfolding** *Prule* **using** *wf-pterm.cases* **by** *force*
from *i* **have** *asi*:*?As*!*i* = σ *?xi*
by *simp*
then have *?As*!*i* ◁ *A*
using *a sigma subst-image-subterm i*′ **by** (*metis is-FunE isfun nth-mem set-vars-term-list to-pterm.simps*(2) *vars-to-pterm*)
moreover from *i subterm*(2) **have** *co-initial* (*?As*!*i*) (*Bs*!*i*)
unfolding *a Prule source.simps source-apply-subst*[*OF to-pterm-wf-pterm*[*of lhs* β]] *source-to-pterm* **using** *l*
by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule comp-def i*′(1) *i*′(2) *length-map nth-map nth-mem set-vars-term-list term-subst-eq-conv*)
moreover have *measure-ov* (*?As*!*i*) (*Bs*!*i*) = 0 **proof**−
{fix *p* **assume** *p*:*p* ∈ *possL* (*?As*!*i*)
let *?pi*=*var-poss-list* (*labeled-source* (*to-pterm* (*lhs* β)))!*i*′
have *pi*:*?pi*=*var-poss-list* (*labeled-lhs* β) !*i*′
by (*simp add*: *var-poss-list-term-lab-to-term*)
have *xi*:*?xi*=*vars-term-list* (*labeled-lhs* β) !*i*′
by (*metis i*′(2) *vars-term-list-labeled-lhs*)
have *xi*′:*?xi*=*vars-term-list* (*labeled-source* (*to-pterm* (*lhs* β))) ! *i*′

```
        using vars-term-list-term-lab-to-term i'(2) by (metis labeled-source-to-term
source-to-pterm)
          have i'l:i' < length (vars-term-list (labeled-lhs β))
            by (simp add: i'(1) vars-term-list-labeled-lhs)
          have i'l':i' < length (vars-term-list (labeled-source (to-pterm (lhs β))))
            by (simp add: i'(1) vars-term-list-term-lab-to-term)
          have (labeled-source (to-pterm (lhs β))) |-?pi = Var ?xi
            using i' using i'l' vars-term-list-var-poss-list xi' by auto
         moreover have possL A = labelposs ((labeled-source (to-pterm (lhs β)))
· (labeled-source ∘ σ))
           using labeled-source-apply-subst to-pterm-wf-pterm unfolding sigma
by metis
          with p have ?pi@p ∈ possL A
            unfolding set-labelposs-subst asi xi' using i'l' by fastforce
          with subterm(5) have ?pi@p ∉ possL B
            by (meson card-eq-0-iff disjoint-iff finite-Int finite-labelposs)
          moreover {assume p ∈ possL (Bs!i)
            then have ?pi@p ∈ {?pi @ q |q. q ∈ labelposs ((⟨map labeled-source
Bs⟩β) ?xi)}
             by (smt (verit) Inl-inject Inr-Inl-False Prule apply-lhs-subst-var-rule i
length-map map-nth-conv mem-Collect-eq subterm.prems(3) term.distinct(1) term.inject(2)
wf-pterm.cases)
            then have ?pi@p ∈ possL B
              unfolding Prule labeled-source.simps set-labelposs-subst xi pi using
i'l by blast
          }
          ultimately have p ∉ possL (Bs!i)
            by blast
        }
        then have possL (?As!i) ∩ possL (Bs!i) = {}
          by blast
        then show ?thesis by simp
      qed
      ultimately have ?As!i ⊥p Bs!i
        using subterm(1,3,4) i unfolding a Prule
           by (smt (verit, best) Inr-Inl-False Term.term.simps(4) length-map
lhs-subst-args-wf-pterm nth-mem sum.inject(1) term.inject(2) wf-pterm.simps)
    }
    then show ?thesis unfolding a Prule using orthogonal.intros(4)[of ?As Bs]
         by (smt (verit, best) Prule Term.term.simps(4) in-set-zip length-map
old.sum.inject(1) prod.case-eq-if subterm.prems(3) sum.distinct(1) term.inject(2)
wf-pterm.cases)
  qed simp
 next
  case (Prule α As)
  then have A:A = Prule α As
    by simp
  from Prule subterm(3) R have lin:linear-term (lhs α)
    unfolding left-lin-no-var-lhs-def left-lin-def left-linear-trs-def
```

using *wf-pterm.simps* **by** *fastforce*
obtain *f ts* **where** *f*:*lhs* α = *Fun f ts*
using *subterm*(*3*) *R no-var-lhs.lhs-is-Fun* **unfolding** *left-lin-no-var-lhs-def Prule* **by** *blast*
show *?thesis* **proof**(*cases B*)
case (*Var x*)
then show *?thesis*
by (*metis source.simps*(*1*) *source-orthogonal subterm.prems*(*1*) *to-pterm.simps*(*1*))
next
case (*Pfun g Bs*)
have (*lhs* α) $\cdot$ (*term-lab-to-term* $\circ$ ($\langle$*map labeled-source As*$\rangle_\alpha$)) = *lhs* α $\cdot$ $\langle$*map source As*$\rangle_\alpha$
by (*metis label-term-to-term labeled-source.simps*(*3*) *labeled-source-to-term source.simps*(*3*) *term-lab-to-term-subst*)
with *subterm*(*2*) **have** *co-init*:*term-lab-to-term* (*labeled-source B*) = *lhs* α $\cdot$ (*term-lab-to-term* $\circ$ $\langle$*map labeled-source As*$\rangle_\alpha$)
unfolding *Prule* **by** *simp*
from *subterm*(*5*) **have** *possL A* $\cap$ *possL B* = {}
by (*simp add*: *finite-possL*)
then obtain τ **where** *labeled-source B* = *term-to-term-lab* (*lhs* α) $\cdot$ τ
unfolding *labeled-source.simps*(*3*) *Prule* **using** *labels-intersect-label-term*[*OF co-init lin*] **by** *blast*
moreover have *labelposs* (*term-to-term-lab* (*lhs* α)) = {}
using *labelposs-term-to-term-lab* **by** *blast*
moreover from *lin* **have** *linear-term* (*term-to-term-lab* (*lhs* α))
using *linear-term-to-term-lab* **by** *auto*
moreover have *term-lab-to-term* (*term-to-term-lab* (*lhs* α)) = *lhs* α
by *simp*
ultimately obtain σ **where** *sigma*:*B* = *to-pterm* (*lhs* α) $\cdot$ σ
using *no-var-lhs.unlabeled-source-to-pterm S subterm*(*4*) **unfolding** *left-lin-no-var-lhs-def* **by** *metis*
let *?Bs*=*map* σ (*var-rule* α)
from *sigma* **have** *b*:*B* = (*to-pterm* (*lhs* α)) $\cdot$ $\langle$*?Bs*$\rangle_\alpha$
by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule comp-apply length-map map-eq-conv set-remdups set-rev set-vars-term-list term-subst-eq vars-to-pterm*)
{**fix** *i* **assume** *i*:*i* < *length* (*var-rule* α)
let *?xi*=*var-rule* α!*i*
from *i* **obtain** *i*′ **where** *i*′:*i*′ < *length* (*vars-term-list* (*lhs* α)) *?xi* = *vars-term-list* (*lhs* α)!*i*′
by (*metis comp-apply in-set-conv-nth set-remdups set-rev*)
from *i* **have** *asi*:*?Bs*!*i* = σ *?xi*
by *simp*
moreover have *l*:*length As* = *length* (*var-rule* α)
using *subterm*(*3*) **unfolding** *A* **using** *wf-pterm.cases* **by** *force*
then have *As*!*i* $\lhd$ *A*
using *i* **unfolding** *A* **by** *simp*
moreover from *i subterm*(*2*) **have** *co-initial* (*As*!*i*) (*?Bs*!*i*)
unfolding *b Prule source.simps source-apply-subst*[*OF to-pterm-wf-pterm*[*of lhs* α]] *source-to-pterm* **using** *l*

by (*smt* (*verit*, *best*) *apply-lhs-subst-var-rule comp-def i'(1) i'(2) length-map nth-map nth-mem set-vars-term-list term-subst-eq-conv*)
moreover have *measure-ov* (*As*!*i*) (*?Bs*!*i*) = *0* **proof**−
{fix *p* **assume** *p*:*p* ∈ *possL* (*?Bs*!*i*)
let *?pi*=*var-poss-list* (*labeled-source* (*to-pterm* (*lhs* α)))!*i'*
have *pi*:*?pi*=*var-poss-list* (*labeled-lhs* α) !*i'*
by (*simp add*: *var-poss-list-term-lab-to-term*)
have *xi*:*?xi*=*vars-term-list* (*labeled-lhs* α) !*i'*
by (*metis i'(2) vars-term-list-labeled-lhs*)
have *xi'*:*?xi*=*vars-term-list* (*labeled-source* (*to-pterm* (*lhs* α))) ! *i'*
using *vars-term-list-term-lab-to-term i'(2)* **by** (*metis labeled-source-to-term source-to-pterm*)
have *i'l*:*i'* < *length* (*vars-term-list* (*labeled-lhs* α))
by (*simp add*: *i'(1) vars-term-list-labeled-lhs*)
have *i'l'*:*i'* < *length* (*vars-term-list* (*labeled-source* (*to-pterm* (*lhs* α))))
by (*simp add*: *i'(1) vars-term-list-term-lab-to-term*)
have (*labeled-source* (*to-pterm* (*lhs* α))) |-*?pi* = *Var ?xi*
using *i'* **using** *i'l' vars-term-list-var-poss-list xi'* **by** *auto*
moreover have *possL B* = *labelposs* ((*labeled-source* (*to-pterm* (*lhs* α))) · (*labeled-source* ∘ σ))
using *labeled-source-apply-subst to-pterm-wf-pterm* **unfolding** *sigma* **by** *metis*
with *p* **have** *?pi*@*p* ∈ *possL B*
unfolding *set-labelposs-subst asi xi'* **using** *i'l'* **by** *fastforce*
with *subterm(5)* **have** *?pi*@*p* ∉ *possL A*
by (*meson card-eq-0-iff disjoint-iff finite-Int finite-labelposs*)
moreover {assume *p* ∈ *possL* (*As*!*i*)
then have *?pi*@*p* ∈ {*?pi* @ *q* |*q*. *q* ∈ *labelposs* ((⟨*map labeled-source As*⟩α) *?xi*)}
by (*smt* (*verit*) *Inl-inject Inr-Inl-False Prule apply-lhs-subst-var-rule i length-map map-nth-conv mem-Collect-eq subterm.prems(2) term.distinct(1) term.inject(2) wf-pterm.cases*)
then have *?pi*@*p* ∈ *possL A*
unfolding *Prule labeled-source.simps set-labelposs-subst xi pi* **using** *i'l* **by** *blast*
}
ultimately have *p* ∉ *possL* (*As*!*i*)
by *blast*
}
then have *possL* (*As*!*i*) ∩ *possL* (*?Bs*!*i*) = {}
by *blast*
then show *?thesis* **by** *simp*
qed
ultimately have *As*!*i* ⊥*p* *?Bs*!*i*
using *subterm(1,3,4) i* **unfolding** *b Prule*
by (*smt* (*verit*, *best*) *Inr-Inl-False Term.term.simps(4) length-map lhs-subst-args-wf-pterm nth-mem sum.inject(1) term.inject(2) wf-pterm.simps*)
}
then show *?thesis* **unfolding** *b Prule* **using** *orthogonal.intros(3)*[*of As ?Bs*]

by (*smt* (*verit*, *best*) *Prule Term.term.simps*(*4*) *in-set-zip length-map old.sum.inject*(*1*) *prod.case-eq-if subterm.prems*(*2*) *sum.distinct*(*1*) *term.inject*(*2*) *wf-pterm.cases*)
next
case (*Prule* β *Bs*)
from *subterm*(*4*) *S* **obtain** *g ss* **where** *g*:*lhs* β = *Fun g ss*
unfolding *Prule left-lin-no-var-lhs-def* **using** *no-var-lhs.lhs-is-Fun* **by** *blast*
have [] ∈ *possL A*
unfolding *A f labeled-source.simps label-term.simps eval-term.simps label-poss.simps* **by** *blast*
moreover have [] ∈ *possL B*
unfolding *Prule g labeled-source.simps label-term.simps eval-term.simps labelposs.simps* **by** *blast*
ultimately show *?thesis*
using *subterm*(*5*) **by** (*simp add*: *disjoint-iff finite-labelposs*)
qed
qed
qed

5.3 Collecting Overlapping Positions

abbreviation *overlaps-pos* :: ('*f*, '*v*) *term-lab* ⇒ ('*f*, '*v*) *term-lab* ⇒ (*pos* × *pos*) *set*
where *overlaps-pos A B* ≡ *Set.filter* (λ(*p*,*q*). *get-label* (*A*|-*p*) ≠ *None* ∧ *get-label* (*B*|-*q*) ≠ *None* ∧
snd (*the* (*get-label* (*A*|-*p*))) = *0* ∧ *snd* (*the* (*get-label* (*B*|-*q*))) = *0* ∧
(*p* <$_p$ *q* ∧ *get-label* (*A*|-*q*) ≠ *None* ∧ *fst* (*the* (*get-label* (*A*|-*q*))) = *fst* (*the* (*get-label* (*A*|-*p*))) ∧ *snd* (*the* (*get-label* (*A*|-*q*))) = *length* (*the* (*remove-prefix p q*)) ∨
(*q* ≤$_p$ *p* ∧ *get-label* (*B*|-*p*) ≠ *None* ∧ *fst* (*the* (*get-label* (*B*|-*q*))) = *fst* (*the* (*get-label* (*B*|-*p*))) ∧ *snd* (*the* (*get-label* (*B*|-*p*))) = *length* (*the* (*remove-prefix q p*)))))
(*fun-poss A* × *fun-poss B*)

lemma *overlaps-pos-symmetric*:
assumes (*p*,*q*) ∈ *overlaps-pos A B*
shows (*q*,*p*) ∈ *overlaps-pos B A*
using *SigmaI assms* **unfolding** *strict-prefix-def* **by** *auto*

lemma *overlaps-pos-intro*:
assumes *q*@*q*′ ∈ *fun-poss A* **and** *q* ∈ *fun-poss B*
and *get-label* (*A*|-(*q*@*q*′)) = *Some* (γ, *0*)
and *get-label* (*B*|-*q*) = *Some* (β, *0*)
and *get-label* (*B*|-(*q*@*q*′)) = *Some* (β, *length q*′)
shows (*q*@*q*′, *q*) ∈ *overlaps-pos A B*
using *assms* **by** *force*

Define the partial order on overlaps

definition *less-eq-overlap* :: *pos* × *pos* ⇒ *pos* × *pos* ⇒ *bool* (**infix** ≤$_o$ *50*)

where $p \leq_o q \longleftrightarrow (fst\ p \leq_p fst\ q) \wedge (snd\ p \leq_p snd\ q)$

definition *less-overlap* :: $pos \times pos \Rightarrow pos \times pos \Rightarrow bool$ (**infix** $<_o$ *50*)
where $p <_o q \longleftrightarrow p \leq_o q \wedge p \neq q$

interpretation *order-overlaps*: *order less-eq-overlap less-overlap*
proof
show $\bigwedge x.\ x \leq_o x$
by (*simp add*: *less-eq-overlap-def*)
show $\bigwedge x\ y\ z.\ x \leq_o y \Longrightarrow y \leq_o z \Longrightarrow x \leq_o z$
using *less-eq-overlap-def* **by** *auto*
show $\bigwedge x\ y.\ (x <_o y) = strict\ (\leq_o)\ x\ y$
using *less-eq-overlap-def less-overlap-def* **by** *fastforce*
thus $\bigwedge x\ y.\ x \leq_o y \Longrightarrow y \leq_o x \Longrightarrow x = y$
by (*meson less-overlap-def*)
qed

lemma *overlaps-pos-finite*: *finite* (*overlaps-pos A B*)
by (*meson finite-SigmaI finite-filter finite-fun-poss*)

lemma *labeled-sources-imp-measure-not-zero*:
assumes $p \in poss$ (*labeled-source A*) $p \in poss$ (*labeled-source B*)
and *get-label* ((*labeled-source A*)|-*p*) $\neq$ *None* $\wedge$ *get-label* ((*labeled-source B*)|-*p*) $\neq$ *None*
shows *measure-ov A B* > *0*
using *assms*
by (*metis card-gt-0-iff disjoint-iff finite-Int finite-possL get-label-imp-labelposs*)

lemma *measure-zero-imp-empty-overlaps*:
assumes *measure-ov A B* = *0* **and** *co-init*:*co-initial A B*
shows *overlaps-pos* (*labeled-source A*) (*labeled-source B*) = {}
using *assms*(*1*) **proof**(*rule contrapos-pp*)
{assume *overlaps-pos* (*labeled-source A*) (*labeled-source B*) $\neq$ {}
then obtain *p q* **where** *pq*:(*p*, *q*) $\in$ *overlaps-pos* (*labeled-source A*) (*labeled-source B*)
by (*meson equals0D pred-equals-eq2*)
then have *get-label* ((*labeled-source A*)|-*p*) $\neq$ *None* $\wedge$ *get-label* ((*labeled-source B*)|-*q*) $\neq$ *None*
$\wedge$ (*get-label* ((*labeled-source A*)|-*q*) $\neq$ *None* $\vee$ *get-label* ((*labeled-source B*)|-*p*) $\neq$ *None*)
by *auto*
moreover from *pq* **have** *p* $\in$ *poss* (*labeled-source A*) **and** *q* $\in$ *poss* (*labeled-source B*)
by (*auto intro*: *fun-poss-imp-poss*)
ultimately show *measure-ov A B* $\neq$ *0*
using *labeled-sources-imp-measure-not-zero co-init*
by (*metis labeled-source-to-term less-numeral-extra*(*3*) *poss-term-lab-to-term*)
}
qed

lemma *empty-overlaps-imp-measure-zero*:
assumes $A \in$ *wf-pterm R* **and** $B \in$ *wf-pterm S*
and *overlaps-pos* (*labeled-source A*) (*labeled-source B*) = {}
shows *measure-ov A B = 0*
using *assms*(*3*) **proof**(*rule contrapos-pp*)
{assume *measure-ov A B* $\neq$ *0*
then obtain *p* **where** *p*:$p \in$ *possL A* $\wedge$ $p \in$ *possL B*
using *Int-emptyI* **by** *force*
then obtain α *n* **where** *a*:*get-label* ((*labeled-source A*)|-*p*) = *Some*(α, *n*)
using *possL-obtain-label* **by** *blast*
let *?p1*=*take* (*length p* − *n*) *p*
obtain *q1* **where** *q1*:*p* = *?p1*@*q1*
by (*metis append-take-drop-id*)
from *a p assms*(*1*) **have** *alpha*:*get-label* (*labeled-source A* |- *?p1*) = *Some* (α, *0*) **and** *?p1* $\in$ *poss* (*labeled-source A*)
using *labelposs-subs-poss obtain-label-root* **by** *blast*+
then have *p1-pos*:*?p1* $\in$ *fun-poss* (*labeled-source A*)
using *get-label-imp-labelposs labelposs-subs-fun-poss* **by** *blast*
from *p* **obtain** β *m* **where** *b*:*get-label* ((*labeled-source B*)|-*p*) = *Some*(β, *m*)
using *possL-obtain-label* **by** *blast*
let *?p2*=*take* (*length p* − *m*) *p*
obtain *q2* **where** *q2*:*p* = *?p2*@*q2*
by (*metis append-take-drop-id*)
from *b p assms*(*2*) **have** *beta*:*get-label* (*labeled-source B* |- *?p2*) = *Some* (β, *0*) **and** *?p2* $\in$ *poss* (*labeled-source B*)
using *labelposs-subs-poss obtain-label-root* **by** *blast*+
then have *p2-pos*:*?p2* $\in$ *fun-poss* (*labeled-source B*)
using *get-label-imp-labelposs labelposs-subs-fun-poss* **by** *blast*

then show *overlaps-pos* (*labeled-source A*) (*labeled-source B*) $\neq$ {} **proof**(*cases* *?p1* $\leq_p$ *?p2*)
case *True*
then obtain *p3* **where** *p2*:*?p2* = *?p1*@*p3*
by (*metis prefix-def*)
with *q2* **have** *p* = *?p1* @ *p3* @ *q2*
by *simp*
with *q1* **have** *p3*:*q1* = *p3*@*q2*
by (*metis same-append-eq*)
from *a alpha q1* **have** *n* = *length q1*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left' append-take-drop-id assms*(*1*) *label-term-max-value labelposs-subs-poss length-drop ordered-cancel-comm-monoid-diff-class.diff-add p same-append-eq subsetD*)
with *p3* **have** *n* = *length p3* + *length q2*
by *auto*
then have *get-label* ((*labeled-source A*)|-(*?p1*@*p3*)) = *Some* (α, *length p3*)
using *label-decrease*[*of ?p1*@*p3 q2 A*] *p1-pos a assms*(*1*)
by (*metis add.commute fun-poss-imp-poss fun-poss-term-lab-to-term labeled-source-to-term labelposs-subs-fun-poss-source p p2 q2*)

```
    then have (?p2, ?p1) ∈ overlaps-pos (labeled-source B) (labeled-source A)
      using overlaps-pos-intro p1-pos p2-pos p2 alpha beta by simp
    then show ?thesis using overlaps-pos-symmetric by blast
  next
    case False
    with q1 q2 have ?p2 <_p ?p1
      by (metis less-eq-pos-simps(1) pos-cases pos-less-eq-append-not-parallel)
    then obtain p3 where p2:?p1 = ?p2@p3
      using less-pos-def' by blast
    with q1 have p = ?p2 @ p3 @ q1
      by simp
    with q2 have p3:q2 = p3@q1
      by (metis same-append-eq)
    from b beta q2 have m = length q2
    by (metis (no-types, lifting) add-diff-cancel-left' append-take-drop-id assms(2)
label-term-max-value labelposs-subs-poss length-drop ordered-cancel-comm-monoid-diff-class.diff-add
p same-append-eq subsetD)
    with p3 have m = length p3 + length q1
      by auto
    then have get-label ((labeled-source B)|-(?p2@p3)) = Some (β, length p3)
      using label-decrease[of ?p2@p3 q1 B] p2-pos b assms(2)
        by (metis add.commute fun-poss-imp-poss fun-poss-term-lab-to-term la-
beled-source-to-term labelposs-subs-fun-poss-source p p2 q1)
    then have (?p1, ?p2) ∈ overlaps-pos (labeled-source A) (labeled-source B)
      using overlaps-pos-intro p1-pos p2-pos p2 alpha beta by simp
    then show ?thesis by blast
  qed
 }
qed

lemma obtain-overlap:
 assumes p ∈ possL A p ∈ possL B
   and get-label (labeled-source A|-p) = Some (γ, n)
   and get-label (labeled-source B|-p) = Some (δ, m)
   and n ≤ length p m ≤ length p
   and rγ = take (length p − n) p
   and rδ = take (length p − m) p
   and rδ ≤_p rγ
   and a-well:A ∈ wf-pterm R and b-well:B ∈ wf-pterm S
 shows (rγ, rδ) ∈ overlaps-pos (labeled-source A) (labeled-source B)
proof−
 from assms(9) obtain r' where r':rγ = rδ @ r'
   using prefix-pos-diff by metis
 have rδ @ r' ∈ fun-poss (labeled-source A)
  using assms(1,7) unfolding r' by (metis append-take-drop-id fun-poss-append-poss'
labelposs-subs-fun-poss subsetD)
 moreover have rδ ∈ fun-poss (labeled-source B)
   using assms(2,4,8) by (metis append-take-drop-id fun-poss-append-poss' label-
poss-subs-fun-poss subsetD)
```

moreover have *get-label* ((*labeled-source A*) |- (*rδ* @ *r′*)) = *Some* (*γ*, *0*)
using *assms*(*1*,*3*,*5*,*7*) *a-well* **unfolding** *r′* **using** *label-decrease*[*of take* (*length p* − *n*) *p drop* (*length p*− *n*) *p*]
by (*smt* (*verit*, *best*) *add.right-neutral add-diff-cancel-left′ append-assoc append-take-drop-id labelposs-subs-poss le-add-diff-inverse2 length-drop subsetD*)
moreover have *get-label* ((*labeled-source B*) |- (*rδ*)) = *Some* (*δ*, *0*)
using *assms*(*2*,*4*,*6*,*8*) *b-well* **using** *label-decrease*[*of take* (*length p* − *m*) *p drop* (*length p*− *m*) *p*]
by (*smt* (*verit*, *best*) *add.right-neutral add-diff-cancel-left′ append-assoc append-take-drop-id labelposs-subs-poss le-add-diff-inverse2 length-drop subsetD*)
moreover have *get-label* ((*labeled-source B*) |- (*rδ*@*r′*)) = *Some* (*δ*, *length r′*)
using *assms*(*2*,*4*,*6*,*8*) *b-well* **unfolding** *r′* **using** *label-decrease*[*of take* (*length p* − *length r′*) *p drop* (*length p*− *length r′*) *p*]
by (*smt* (*verit*, *del-insts*) *Nat.add-diff-assoc add-diff-cancel-left′ append.assoc append-take-drop-id assms*(*7*) *diff-diff-cancel diff-le-self fun-poss-imp-poss fun-poss-term-lab-to-term label-decrease labeled-source-to-term labelposs-subs-fun-poss-source le-add1 le-add-diff-inverse length-append length-take min.absorb2 r′*)
ultimately show *?thesis* **using** *overlaps-pos-intro* **unfolding** *r′*
by *auto*
qed

end

6 Redex Patterns

theory *Redex-Patterns*
imports
Labels-and-Overlaps
begin

Collect all rule symbols of a proof term together with the position in its source where they appear. This is used to split a proof term into a set of single steps, whose union (⨆) is the whole proof term again.

The redex patterns are collected in leftmost outermost order.

fun *redex-patterns* :: (*'f*, *'v*) *pterm* ⇒ ((*'f*, *'v*) *prule* × *pos*) *list*
where
redex-patterns (*Var x*) = []
| *redex-patterns* (*Pfun f ss*) = *concat* (*map* (λ (*i*, *rps*). *map* (λ (*α*, *p*). (*α*, *i*#*p*)) *rps*)
(*zip* [*0* ..< *length ss*] (*map redex-patterns ss*)))
| *redex-patterns* (*Prule α ss*) = (*α*, []) # *concat* (*map* (λ (*p1*, *rps*). *map* (λ (*α*, *p2*). (*α*, *p1*@*p2*)) *rps*)
(*zip* (*var-poss-list* (*lhs α*)) (*map redex-patterns ss*)))

interpretation *lexord-linorder*:
linorder ord.lexordp-eq ((<) :: *nat* ⇒ *nat* ⇒ *bool*)
ord.lexordp ((<) :: *nat* ⇒ *nat* ⇒ *bool*)
using *linorder.lexordp-linorder*[*OF linorder-class.linorder-axioms*] **by** *simp*

```
lemma lexord-prefix-diff:
  assumes (ord.lexordp ((<) :: nat ⇒ nat ⇒ bool)) xs ys and ¬ prefix xs ys
  shows (ord.lexordp (<)) (xs@us) (ys@vs)
using assms proof(induct xs arbitrary:ys)
  case (Cons x xs')
  from Cons(2) obtain y ys' where ys:ys = y#ys'
    by (metis list.exhaust-sel ord.lexordp-simps(2))
  consider x < y | x = y ∧ ord.lexordp (<) xs' ys'
    using Cons(2) ord.lexordp-eq.simps unfolding Cons ys by force
  then show ?case proof(cases)
    case 1
    then show ?thesis unfolding ys by simp
  next
    case 2
    with Cons(3) have ¬ prefix xs' ys'
      unfolding ys by simp
    with Cons(1) 2 have (ord.lexordp (<)) (xs'@us) (ys'@vs)
      by auto
    then show ?thesis unfolding ys using 2 by simp
  qed
qed simp

lemma var-poss-list-sorted: sorted-wrt (ord.lexordp ((<) :: nat ⇒ nat ⇒ bool))
(var-poss-list t)
proof(induct t)
  case (Fun f ts)
  let ?poss=(map2 (λi. map ((#) i)) [0..<length ts] (map var-poss-list ts))
  {fix i j assume i:i < length (var-poss-list (Fun f ts)) and j:j < i
    let ?p=concat ?poss ! i
    let ?q=concat ?poss ! j
    from i obtain i' i'' where p:?p = ?poss!i'!i'' and i':i' < length ts and i'':i''
< length (?poss!i')
        and i-sum:i = sum-list (map length (take i' ?poss)) + i''
     using less-length-concat[OF i[unfolded var-poss-list.simps]] unfolding length-map
by auto
    from p have p2:?p = i'# (var-poss-list (ts!i') ! i'')
      using i' i'' by simp
    from j obtain j' j'' where q:?q = ?poss!j'!j'' and j':j' < length ts and j'':j''
< length (?poss!j')
        and j-sum:j = sum-list (map length (take j' ?poss)) + j''
      using less-length-concat i j length-map unfolding var-poss-list.simps
        by (smt (verit, ccfv-threshold) diff-le-self length-take length-upt length-zip
map-upt-len-conv order.strict-trans take-all)
    from q have q2:?q = j'# (var-poss-list (ts!j') ! j'')
      using j' j'' by simp
    have (ord.lexordp (<)) (var-poss-list (Fun f ts) ! j) (var-poss-list (Fun f ts) ! i)
proof(cases i' = j')
      case True
```

```
    have l:length (map2 (λx. map ((#) x)) [0..<length ts] (map var-poss-list ts)
! j') = length (var-poss-list (ts!j'))
      using j' by auto
    from True j j-sum i-sum have j'' < i''
      using nat-add-left-cancel-less by blast
     with Fun(1)[of ts!j'] i' i'' j'' have (ord.lexordp (<)) (var-poss-list (ts!j') !
j'') (var-poss-list (ts!i') ! i'')
      unfolding True l by (simp add: sorted-wrt-iff-nth-less)
    then have (ord.lexordp (<)) ?q ?p
      unfolding p2 q2 True by simp
    then show ?thesis unfolding var-poss-list.simps by fastforce
   next
    case False
    then have j' < i'
      using  i'' i' j' i-sum j-sum sum-list-less[OF j]
     by (smt (verit, best) i j le-neq-implies-less length-concat linorder-le-less-linear
not-add-less1 order.strict-trans take-all var-poss-list.simps(2))
    then have (ord.lexordp (<)) ?q ?p
      unfolding p2 q2 by simp
    then show ?thesis unfolding var-poss-list.simps by fastforce
   qed
  }
  then show ?case
   using sorted-wrt-iff-nth-less by blast
qed simp

context left-lin-no-var-lhs
begin

lemma redex-patterns-sorted:
  assumes  A ∈ wf-pterm R
  shows sorted-wrt (ord.lexordp (<)) (map snd (redex-patterns A))
proof−
  {fix i j assume i < j j < length (redex-patterns A)
   with assms have (ord.lexordp (<)) (snd (redex-patterns A ! i)) (snd (redex-patterns
A ! j))
    proof(induct A arbitrary: i j)
     case (2 As f)
      let ?poss=map2 (λi. map (λ(α, p). (α, i # p))) [0..<length As] (map re-
dex-patterns As)
     from 2(2,3) obtain α p1 where ap:redex-patterns (Pfun f As) ! i = (α, p1)
       by (metis surj-pair)
     from 2(3) obtain β p2 where bp:redex-patterns (Pfun f As) ! j = (β, p2)
       by (metis surj-pair)
      have l:length (zip [0..<length As] (map redex-patterns As)) = length As by
simp
     from 2(2,3) have ∗:i < length (concat ?poss) by simp
     from ap obtain i' i'' where ap1:(α, p1) = ?poss!i'!i'' and i':i' < length As
and i'':i'' < length (?poss!i')
```

and *i*:*i* = *sum-list* (*map length* (*take i′ ?poss*)) + *i″*
unfolding *redex-patterns.simps* **using** *less-length-concat*[*OF* ∗] **by** (*metis l length-map*)
have *poss-i′*:*?poss*!*i′* = *map* (λ(α, *p*). (α, *i′* # *p*)) (*redex-patterns* (*As*!*i′*))
using *i′ nth-zip*[*of i′*] **by** *fastforce*
from *ap1 i″* **obtain** *p1′* **where** *p1′*:*p1* = *i′*#*p1′* (α, *p1′*) = *redex-patterns* (*As*!*i′*) ! *i″*
unfolding *poss-i′* **by** (*smt* (*z3*) *case-prod-conv length-map nth-map old.prod.inject surj-pair*)
from *bp* **obtain** *j′ j″* **where** *ap2*:(β, *p2*) = *?poss*!*j′*!*j″* **and** *j′*:*j′* < *length As* **and** *j″*:*j″* < *length* (*?poss*!*j′*)
and *j*:*j* = *sum-list* (*map length* (*take j′ ?poss*)) + *j″*
unfolding *redex-patterns.simps* **using** *less-length-concat*[*OF 2*(*3*)[*unfolded redex-patterns.simps*]] **by** (*metis l length-map*)
have *poss-j′*:*?poss*!*j′* = *map* (λ(α, *p*). (α, *j′* # *p*)) (*redex-patterns* (*As*!*j′*))
using *j′ nth-zip*[*of j′*] **by** *fastforce*
from *ap2 j″* **obtain** *p2′* **where** *p2′*:*p2* = *j′*#*p2′* (β, *p2′*) = *redex-patterns* (*As*!*j′*) ! *j″*
unfolding *poss-j′* **by** (*smt* (*z3*) *case-prod-conv length-map nth-map old.prod.inject surj-pair*)
show *?case* **proof**(*cases i′* = *j′*)
case *True*
from *i j 2* **have** *i″* < *j″* **unfolding** *True* **by** *linarith*
moreover from *j″* **have** *j″* < *length* (*redex-patterns* (*As*!*j′*)) **unfolding** *poss-j′* **by** *auto*
ultimately have *ord.lexordp* (<) *p1′ p2′* **using** *2*(*1*) *j′ True p1′*(*2*) *p2′*(*2*) **by** (*metis nth-mem snd-eqD*)
then show *?thesis* **unfolding** *ap bp p1′ p2′ True* **by** *auto*
next
case *False*
with *2*(*2*) *i j* **have** *i′* < *j′* **using** *sum-list-less*[*OF 2*(*2*)] *i′ j′ j″*
by (*smt* (*verit*, *best*) ∗ *2.prems*(*2*) *le-neq-implies-less length-concat linorder-le-less-linear not-add-less1 redex-patterns.simps*(*2*) *take-all*)
then show *?thesis* **unfolding** *ap bp p1′ p2′* **by** *fastforce*
qed
next
case (*3* γ *As*)
from *3*(*2*,*3*) **obtain** α *p1* **where** *ap*:*redex-patterns* (*Prule* γ *As*) ! *i* = (α, *p1*)
by (*metis surj-pair*)
from *3*(*3*) **obtain** β *p2* **where** *bp*:*redex-patterns* (*Prule* γ *As*) ! *j* = (β, *p2*)
by (*metis surj-pair*)
show *?case* **proof**(*cases i*)
case *0*
from *3*(*1*) *no-var-lhs* **obtain** *f ts* **where** *lhs*:*lhs* γ = *Fun f ts*
by *fastforce*
from *bp 3*(*4*) *0* **obtain** *j′* **where** *concat* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*)) ! *j′* = (β, *p2*)
j′ < *length* (*concat* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list*

(*lhs* γ)) (*map redex-patterns As*)))
unfolding *redex-patterns.simps* **using** *3.prems*(*2*) **by** *force*
then obtain *j1 j2* **where** *j1*:*j1* < *length* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*))
and *j2*:*j2* < *length* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*) ! *j1*)
and *j1j2*:(*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*) ! *j1*) ! *j2* = (β, *p2*)
using *nth-concat-split* **by** *metis*
let *?p'*=*var-poss-list* (*lhs* γ)!*j1*
let *?rdp*=(*map redex-patterns As* ! *j1*)
from *j1* **have** *zip*:*zip* (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*) ! *j1* = (*?p'*, *?rdp*)
unfolding *length-map length-zip* **using** *nth-zip* **by** *force*
with *j1j2* **have** (β, *p2*) = *map* (λ(α, *p2*). (α, *?p'* @ *p2*)) *?rdp* !*j2*
using *nth-map j1* **unfolding** *length-map* **by** *force*
moreover from *j2* **have** *j2* < *length* (*map* (λ(α, *p2*). (α, *?p'* @ *p2*)) *?rdp*)
unfolding *nth-map*[*OF j1*[*unfolded length-map*]] *zip* **by** *force*
ultimately have (β, *p2*) ∈ *set* (*map* (λ(α, *p2*). (α, *?p'* @ *p2*)) *?rdp*)
by *simp*
moreover have *?p'* ≠ [] **proof**−
from *j1* **have** *?p'* ∈ *var-poss* (*lhs* γ)
unfolding *length-map length-zip* **using** *nth-mem* **by** *fastforce*
then show *?thesis* **unfolding** *lhs var-poss.simps* **by** *force*
qed
ultimately have *p2* ≠ []
by *auto*
moreover from *0 ap* **have** *p1* = [] **by** *simp*
ultimately show *?thesis* **unfolding** *ap bp* **by** *simp*
next
case (*Suc n*)
let *?poss*=(*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*))
from *3*(*1*,*2*) **have** *l*:*length* (*var-poss-list* (*lhs* γ)) = *length As*
using *linear-term-var-vars-term-list left-lin* **unfolding** *left-linear-trs-def* **using** *length-var-poss-list length-var-rule* **by** *auto*
from *3*(*4*,*5*) **have** ∗:*n* < *length* (*concat ?poss*) **unfolding** *Suc* **by** *simp*
from *ap* **have** *concat* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* γ)) (*map redex-patterns As*)) ! *n* = (α, *p1*)
unfolding *Suc* **by** *simp*
then obtain *i' i''* **where** *ap1*:(α, *p1*) = *?poss*!*i'*!*i''* **and** *i'*:*i'* < *length ?poss* **and** *i''*:*i''* < *length* (*?poss*!*i'*)
and *n*:*n* = *sum-list* (*map length* (*take i' ?poss*)) + *i''*
using *less-length-concat*[*OF* ∗] **by** *metis*
from *i'* **have** *i'2*:*i'* < *length* (*var-poss-list* (*lhs* γ)) **by** *simp*
obtain *p11* **where** *p11*:*?poss*!*i'* = *map* (λ(α, *p*). (α, *p11* @ *p*)) (*redex-patterns* (*As*!*i'*)) *var-poss-list* (*lhs* γ) !*i'* = *p11*
using *i' nth-zip*[*of i'*] **by** *fastforce*

```
    from ap1 i'' obtain p12 where p12:p1 = p11@p12 (α, p12) = redex-patterns
(As!i') ! i''
      unfolding p11 by (smt (z3) case-prod-conv length-map nth-map old.prod.inject
surj-pair)
       from 3(4,5) Suc obtain n' where j:j = Suc n' by (meson Suc-lessE)
       from 3(5) have *:n' < length (concat ?poss) unfolding j by simp
          from bp have concat (map2 (λp1. map (λ(α, p2). (α, p1 @ p2)))
(var-poss-list (lhs γ)) (map redex-patterns As)) ! n' = (β, p2)
        unfolding j by simp
      then obtain j' j'' where ap2:(β, p2) = ?poss!j'!j'' and j':j' < length ?poss
and j'':j'' < length (?poss!j')
        and n':n' = sum-list (map length (take j' ?poss)) + j''
        using less-length-concat[OF *] by metis
       from j' have j'2:j' < length (var-poss-list (lhs γ)) by simp
     obtain p21 where p21:?poss!j' = map (λ(α, p). (α, p21 @ p)) (redex-patterns
(As!j')) var-poss-list (lhs γ) !j' = p21
        using j' nth-zip[of j'] by fastforce
     from ap2 j'' obtain p22 where p22:p2 = p21@p22 (β, p22) = redex-patterns
(As!j') ! j''
      unfolding p21 by (smt (z3) case-prod-conv length-map nth-map old.prod.inject
surj-pair)
       show ?thesis proof(cases i' = j')
        case True
        from n n' 3(4) have ij:i'' < j'' unfolding True Suc j by linarith
         moreover from j'' have j'' < length (redex-patterns (As!j')) unfolding
p21 by auto
        moreover from j' l have j' < length As unfolding length-map by simp
       ultimately have ord.lexordp (<) p12 p22 using 3(3) p22 p12 j' unfolding
True by (metis nth-mem snd-conv)
          with p21(2) p11(2) show ?thesis unfolding ap bp p22 p12 True by
(simp add: ord.lexordp-append-leftI)
       next
        case False
        then have i' < j'
          using  sum-list-less[OF 3(4), where i'=i' and j'=j']
         by (smt (verit) 3.prems(1) Suc Suc-less-SucD i' j j'' le-neq-implies-less
n n' sum-list-less)
        then have ord.lexordp (<) p11 p21
              using p11(2) p21(2) var-poss-list-sorted[of lhs γ] i'2 j'2 using
sorted-wrt-nth-less by blast
        moreover have ¬ prefix p11 p21 proof−
          from False j' i' have parallel-pos (var-poss-list (lhs γ) !i') (var-poss-list
(lhs γ) !j')
        unfolding length-map length-zip using var-poss-parallel var-poss-list-sound
distinct-var-poss-list
            by (metis l min.idem nth-eq-iff-index-eq nth-mem)
          then show ?thesis using p11(2) p21(2)
            by (metis less-eq-pos-simps(1) parallel-pos prefix-def)
        qed
```

```
      ultimately show ?thesis unfolding ap bp p12 p22 using lexord-prefix-diff
by simp
       qed
     qed
   qed simp
  }
  then show ?thesis
    by (metis (mono-tags, lifting) sorted-wrt-iff-nth-less sorted-wrt-map)
qed

corollary distinct-snd-rdp:
  assumes A ∈ wf-pterm R
  shows distinct (map snd (redex-patterns A))
  using redex-patterns-sorted[OF assms] lexord-linorder.strict-sorted-iff by simp

lemma redex-patterns-equal:
  assumes wf:A ∈ wf-pterm R
     and sorted:sorted-wrt (ord.lexordp (<)) (map snd xs) and eq:set xs = set
(redex-patterns A)
  shows xs = redex-patterns A
proof−
  have linord:class.linorder (ord.lexordp-eq ((<) :: nat ⇒ nat ⇒ bool)) (ord.lexordp
(<))
    using linorder.lexordp-linorder[OF linorder-class.linorder-axioms] by simp
  then have map snd xs = map snd (redex-patterns A)
   using linorder.strict-sorted-equal[OF linord redex-patterns-sorted[OF wf] sorted]
eq by simp
  with eq distinct-snd-rdp[OF wf] show ?thesis
   using distinct-map by (metis (mono-tags, opaque-lifting) inj-onD list.inj-map-strong)

qed

lemma redex-patterns-order:
  assumes A ∈ wf-pterm R and i < j and j < length (redex-patterns A)
    and redex-patterns A ! i = (α, p1) and redex-patterns A ! j = (β, p2)
  shows ¬ p2 ≤p p1
proof−
  have (ord.lexordp (<)) p1 p2
   using redex-patterns-sorted[OF assms(1)] assms sorted-wrt-nth-less by fastforce

  then show ?thesis
    by (metis prefix-def lexord-linorder.less-le-not-le ord.lexordp-eq-pref)
qed

end

context left-lin-no-var-lhs
begin
lemma redex-patterns-label:
```

assumes $A \in$ *wf-pterm R*
shows $(\alpha,\ p) \in$ *set* (*redex-patterns A*) $\longleftrightarrow$ $p \in$ *poss* (*source A*) $\wedge$ *get-label* (*labeled-source A* |- *p*) = *Some* (α, *0*)
proof
{assume $(\alpha,\ p) \in$ *set* (*redex-patterns A*)
with *assms* **show** $p \in$ *poss* (*source A*) $\wedge$ *get-label* (*labeled-source A* |- *p*) = *Some* (α, *0*) **proof**(*induct arbitrary*:*p*)
case (*2 ts f*)
have *l*:*length* (*map2* ($\lambda i.$ *map* ($\lambda(\alpha,\ p).$ (α, *i* # *p*))) [*0*..<*length ts*] (*map redex-patterns ts*)) = *length ts*
unfolding *length-map length-zip* **by** *simp*
with *2*(*2*) **obtain** *i* **where** *i*:*i* < *length ts* **and** *ap*:(α, *p*) $\in$ *set* ((*map2* ($\lambda i.$ *map* ($\lambda(\alpha,\ p).$ (α, *i* # *p*))) [*0*..<*length ts*] (*map redex-patterns ts*))!*i*)
unfolding *redex-patterns.simps* **using** *in-set-idx* **by** (*metis nth-concat-split nth-mem*)
have (*map2* ($\lambda i.$ *map* ($\lambda(\alpha,\ p).$ (α, *i* # *p*))) [*0*..<*length ts*] (*map redex-patterns ts*))!*i* = *map* ($\lambda(\alpha,\ p).$ (α, *i* # *p*)) (*redex-patterns* (*ts*!*i*))
using *nth-zip i* **by** *fastforce*
with *ap* **obtain** *p′* **where** *p′*:*p* = *i*#*p′* **and** (α, *p′*) $\in$ *set* (*redex-patterns* (*ts*!*i*)) **by** *auto*
with *2*(*1*) *i* **have** *p′* $\in$ *poss* (*source* (*ts*!*i*)) **and** *get-label* (*labeled-source* (*ts*!*i*)|-*p′*) = *Some* (α, *0*)
using *nth-mem* **by** *blast+*
with *i* **show** *?case* **unfolding** *p′* **by** *simp*
next
case (*3* β *As*)
from *no-var-lhs 3*(*1*) **obtain** *f ts* **where** *lhs*:*lhs* β = *Fun f ts* **by** *fastforce*
from *3*(*2*) **have** *l*:*length* (*var-poss-list* (*lhs* β)) = *length As*
using *left-lin*.*length-var-rule*[*OF left-lin-axioms 3*(*1*)] **by** (*simp add*: *length-var-poss-list*)

from *3*(*4*) **consider** (*root*) (α, *p*) = (β, []) | (*arg*) (α, *p*) $\in$ *set* (*concat* (*map2* ($\lambda p1.$ *map* ($\lambda(\alpha,\ p2).$ (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*)))
unfolding *redex-patterns.simps* **by** (*meson set-ConsD*)
then show *?case* **proof**(*cases*)
case *root*
then have $\alpha = \beta$ **and** *p* = [] **by** *simp+*
then show *?thesis* **by** (*simp add*: *lhs*)
next
case *arg*
then obtain *i* **where** *i*:*i* < *length As* **and** *ap*:(α, *p*) $\in$ *set* ((*map2* ($\lambda p1.$ *map* ($\lambda(\alpha,\ p2).$ (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*))!*i*)
using *in-set-idx l* **by** (*metis* (*no-types*, *lifting*) *length-map map-snd-zip nth-concat-split nth-mem*)
let *?p1*=(*var-poss-list* (*lhs* β))!*i*
have (*map2* ($\lambda p1.$ *map* ($\lambda(\alpha,\ p2).$ (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*))!*i* = *map* ($\lambda(\alpha,\ p).$ (α, *?p1* @ *p*)) (*redex-patterns* (*As*!*i*))
using *nth-zip i l* **by** *fastforce*
with *ap* **obtain** *p2* **where** *p2*:*p* = *?p1*@*p2* **and** *ap2*:(α, *p2*) $\in$ *set*

(*redex-patterns* (*As* !*i*)) **by** *auto*
with *3*(*3*) *i* **have** *poss*:*p2* ∈ *poss* (*source* (*As*!*i*)) **and** *label*:*get-label* (*labeled-source* (*As*!*i*)|-*p2*) = *Some* (α, *0*)
using *nth-mem* **by** *blast*+
have *p1-poss*:*?p1* ∈ *poss* (*lhs* β) **using** *i l*
by (*metis nth-mem var-poss-imp-poss var-poss-list-sound*)
then have *1*:*p* ∈ *poss* (*source* (*Prule* β *As*))
using *poss 3*(*2*) *i l* **unfolding** *source.simps p2*
by (*smt* (*verit*, *ccfv-SIG*) *append-eq-append-conv2 comp-apply length-map length-remdups-eq length-rev length-var-poss-list nth-map poss-append-poss poss-imp-subst-poss rev-swap var-rule-pos-subst vars-term-list-var-poss-list*)
have *labeled-source* (*Prule* β *As*) |-*p* = *labeled-source* (*As*!*i*) |-*p2* **proof**−
have (⟨*map labeled-source As*⟩$_\beta$) (*var-rule* β ! *i*) = *labeled-source* (*As*!*i*)
using *i 3*(*2*) **by** (*metis length-map lhs-subst-var-i nth-map*)
moreover have *labeled-lhs* β |- *?p1* = *Var* (*var-rule* β ! *i*)
using *3*(*1*) *i l* **by** (*metis case-prodD left-lin left-linear-trs-def length-var-poss-list linear-term-var-vars-term-list p1-poss var-label-term vars-term-list-var-poss-list*)
ultimately show *?thesis* **unfolding** *p2 labeled-source.simps*
by (*smt* (*verit*, *best*) *eval-term.simps*(*1*) *label-term-to-term p1-poss poss-imp-subst-poss poss-term-lab-to-term subt-at-append subt-at-subst*)
qed
with *label* **have** *2*:*get-label* (*labeled-source* (*Prule* β *As*)|-*p*) = *Some* (α, *0*)
by *presburger*
from *1 2* **show** *?thesis* **by** *simp*
qed
qed *simp*
}
{assume *p* ∈ *poss* (*source A*) ∧ *get-label* (*labeled-source A* |- *p*) = *Some* (α, *0*)
with *assms* **show** (α, *p*) ∈ *set* (*redex-patterns A*) **proof**(*induct arbitrary*:*p*)
case (*2 ts f*)
from *2*(*2*) **have** *p* ≠ [] **unfolding** *labeled-source.simps* **by** *auto*
with *2*(*2*) **obtain** *i p′* **where** *p′*:*p* = *i*#*p′ p′* ∈ *poss* (*source* (*ts*!*i*)) **and** *i*:*i* < *length ts*
unfolding *source.simps* **by** *fastforce*
with *2*(*2*) **have** *get-label* (*labeled-source* (*ts*!*i*) |- *p′*) = *Some* (α, *0*)
unfolding *p′ labeled-source.simps* **by** *auto*
with *2*(*1*) *i p′* **have** *IH*:(α, *p′*) ∈ *set* (*redex-patterns* (*ts*!*i*))
using *nth-mem* **by** *blast*
from *i* **have** *i-zip*:*i* < *length* (*zip* [*0*..<*length ts*] (*map redex-patterns ts*)) **by** *simp*
from *i* **have** *zip* [*0*..<*length ts*] (*map redex-patterns ts*) ! *i* = (*i*, *redex-patterns* (*ts*!*i*))
using *nth-zip* **by** *simp*
then have (*map2* (λ*x*. *map* (λ(α, *p*). (α, *x* # *p*))) [*0*..<*length ts*] (*map redex-patterns ts*)) ! *i* = *map* (λ(α, *p*). (α, *i* # *p*)) (*redex-patterns* (*ts*!*i*))
unfolding *nth-map*[*OF i-zip*] **by** *simp*
with *p′*(*2*) *IH* **have** (α, *p*) ∈ *set* ((*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length ts*] (*map redex-patterns ts*))!*i*)
unfolding *p′* **by** *auto*

with *i-zip* **show** *?case* **using** *i* **unfolding** *redex-patterns.simps set-concat* **by** (*metis* (*no-types*, *lifting*) *UN-I length-map nth-mem*)
next
case (*3* β *As*)
with *get-label-Prule* **consider** (*1*)$p = [] \wedge \alpha = \beta$ | ($\exists$ *p1 p2 i*. $p = p1 \ @ \ p2 \wedge i <$ *length As* $\wedge$ *var-poss-list* (*lhs* β) ! $i = p1 \wedge p2 \in$ *poss* (*source* (*As* ! *i*)) $\wedge$ *get-label* (*labeled-source* (*As* ! *i*) |- *p2*) = *Some* (α, *0*))
by (*metis wf-pterm.intros*(*3*))
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **unfolding** *redex-patterns.simps* **by** *simp*
next
case *2*
from *3*(*1*,*2*) *left-lin* **have** *l*:*length* (*var-poss-list* (*lhs* β)) = *length As*
using *length-var-poss-list length-var-rule* **by** *auto*
from *2* **obtain** *p1 p2 i* **where** *p*:$p = p1 \ @ \ p2$ **and** *i*:$i <$ *length As* **and** *p1*:*var-poss-list* (*lhs* β) ! $i = p1$
and *p2*:$p2 \in$ *poss* (*source* (*As* ! *i*)) **and** *lab*:*get-label* (*labeled-source* (*As* ! *i*) |- *p2*) = *Some* (α, *0*)
by *blast*
from *i l* **have** *i'*:$i <$ *length* (*zip* (*var-poss-list* (*lhs* β)) (*map redex-patterns As*)) **by** *simp*
from *i p2 lab 3*(*3*) **have** (α, *p2*) $\in$ *set* (*redex-patterns* (*As*!*i*)) **using** *nth-mem* **by** *blast*
then have (α, *p*) $\in$ *set* (*map* (λ(α, *p2*). (α, *p1* @ *p2*)) (*redex-patterns* (*As*!*i*))) **using** *p* **by** *force*
then have (α, *p*) $\in$ *set* ((*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*))!*i*)
unfolding *nth-map*[*OF i'*] *p* **using** *p1* **by** (*simp add*: *i l*)
then have (α, *p*) $\in$ *set* (*concat* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*)))
unfolding *set-concat* **by** (*metis* (*no-types*, *lifting*) *UN-I i' length-map nth-mem*)
then show *?thesis* **unfolding** *redex-patterns.simps* **by** (*meson list.set-intros*(*2*))
qed
qed *simp*
}
qed

lemma *redex-pattern-rule-symbol*:
assumes *A* $\in$ *wf-pterm R* (α, *p*) $\in$ *set* (*redex-patterns A*)
shows *to-rule* $\alpha \in R$
proof−
from *redex-patterns-label*[*OF assms*(*1*)] **have** *p* $\in$ *poss* (*source A*) **and** *get-label* (*labeled-source A* |- *p*) = *Some* (α, *0*)
using *assms*(*2*) **by** *simp*+
then show *?thesis*
using *assms*(*1*) *labeled-wf-pterm-rule-in-TRS* **by** *fastforce*

qed

lemma *redex-patterns-subset-possL*:
assumes $A \in$ *wf-pterm R*
shows *set* (*map snd* (*redex-patterns A*)) $\subseteq$ *possL A*
using *redex-patterns-label*[*OF assms*]
by (*smt* (*verit*) *get-label-imp-labelposs imageE labeled-source-to-term list.set-map option.simps*(*3*) *poss-term-lab-to-term prod.collapse subsetI*)
end

lemma *redex-poss-empty-imp-empty-step*:
assumes *redex-patterns A* = []
shows *is-empty-step A*
using *assms* **proof**(*induct A*)
case (*Pfun f As*)
{fix *i* **assume** *i*:*i* < *length As*
then have *i-zip*:*i* < *length* (*zip* [*0*..<*length As*] (*map redex-patterns As*)) **by** *simp*
{fix *x xs* **assume** *redex-patterns* (*As*!*i*) = *x*#*xs*
with *i* **have** *zip* [*0*..<*length As*] (*map redex-patterns As*) ! *i* = (*i*, *x*#*xs*) **by** *simp*
then have (*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length As*] (*map redex-patterns As*))!*i* ≠ []
using *nth-map i-zip* **by** *simp*
with *Pfun*(*2*) **have** *False* **unfolding** *redex-patterns.simps* **using** *i-zip concat-nth-length*
by (*metis* (*no-types*, *lifting*) *length-0-conv length-greater-0-conv length-map less-nat-zero-code*)
}
then have *redex-patterns* (*As*!*i*) = []
by (*meson list.exhaust*)
with *Pfun*(*1*) *i* **have** *is-empty-step* (*As*!*i*)
by *simp*
}
then show *?case*
by (*simp add*: *list-all-length*)
qed *simp-all*

lemma *overlap-imp-redex-poss*:
assumes $A \in$ *wf-pterm R* $B \in$ *wf-pterm R*
and *measure-ov A B* ≠ *0*
shows *redex-patterns A* ≠ []
proof
assume *redex-patterns A* = []
then have *is-empty-step A*
by (*simp add*: *redex-poss-empty-imp-empty-step*)
with *assms*(*3*) **show** *False*
by (*simp add*: *empty-step-imp-measure-zero*)
qed

lemma *redex-patterns-to-pterm*:
shows *redex-patterns* (*to-pterm s*) = []
proof(*induct s*)
case (*Fun f ts*)
have *l*:*length* (*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length* (*map to-pterm ts*)] (*map redex-patterns* (*map to-pterm ts*))) = *length ts*
by *simp*
{fix *i* **assume** *i* < *length ts*
with *Fun* **have** (*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length* (*map to-pterm ts*)] (*map redex-patterns* (*map to-pterm ts*)))!*i* = []
by *simp*
}
with *l* **show** *?case* **unfolding** *to-pterm.simps redex-patterns.simps*
by (*metis length-greater-0-conv length-nth-simps*(*1*) *less-nat-zero-code nth-concat-split*)
qed *simp*

lemma *redex-patterns-elem-fun*:
assumes (α, *p*) ∈ *set* (*redex-patterns* (*Pfun f As*))
shows ∃ *i p*′. *i* < *length As* ∧ *p* = *i*#*p*′ ∧ (α, *p*′) ∈ *set* (*redex-patterns* (*As*!*i*))
proof−
let *?xs*=*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length As*] (*map redex-patterns As*)
from *assms* **obtain** *k* **where** *k*:*k* < *length* (*redex-patterns* (*Pfun f As*)) *redex-patterns* (*Pfun f As*) ! *k* = (α, *p*)
by (*metis in-set-idx*)
then obtain *i j* **where** *i* < *length ?xs* **and** *j*:*j* < *length* (*?xs*!*i*) *?xs* ! *i* ! *j* = (α, *p*)
using *nth-concat-split*[*OF k*(*1*)[*unfolded redex-patterns.simps*]] **by** *force*
then have *i*:*i* < *length As* **by** *auto*
then have *zip* [*0*..<*length As*] (*map redex-patterns As*) !*i* = (*i*, *redex-patterns* (*As*!*i*))
using *nth-zip* **by** *auto*
then have *?xs*!*i* = *map* (λ(α, *p*). (α, *i*#*p*)) (*redex-patterns* (*As*!*i*)) **using** *nth-map i* **by** *auto*
with *j* **obtain** *p*′ **where** *p* = *i*#*p*′ **and** (α, *p*′) ∈ *set* (*redex-patterns* (*As*!*i*))
by (*smt* (*verit*, *ccfv-threshold*) *case-prod-beta fst-conv imageE list.set-map nth-mem prod.collapse snd-conv*)
with *i* **show** *?thesis* **by** *simp*
qed

lemma *redex-patterns-elem-rule*:
assumes (α, *p*) ∈ *set* (*redex-patterns* (*Prule* β *As*))
shows *p* = [] ∨ (∃ *i p*′. *i* < *length As* ∧ *i* < *length* (*var-poss-list* (*lhs* β))
∧ *p* = (*var-poss-list* (*lhs* β)!*i*)@*p*′ ∧ (α, *p*′) ∈ *set* (*redex-patterns* (*As*!*i*)))
proof−
let *?xs*=*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* β)) (*map redex-patterns As*)

from *assms* **obtain** *k* **where** *k*:*k* < *length* (*redex-patterns* (*Prule* β *As*)) *redex-patterns* (*Prule* β *As*) ! *k* = (α, *p*)
by (*metis in-set-idx*)
show *?thesis* **proof**(*cases p* = [])
case *False*
with *k* **have** *k* ≠ *0*
unfolding *redex-patterns.simps* **by** (*metis nth-Cons-0 prod.inject*)
with *k* **obtain** *i j* **where** *i* < *length ?xs* **and** *j*:*j* < *length* (*?xs*!*i*) *?xs* ! *i* ! *j* = (α, *p*)
using *nth-concat-split less-Suc-eq-0-disj* **unfolding** *redex-patterns.simps* **by** *force*
then have *i*:*i* < *length As i* < *length* (*var-poss-list* (*lhs* β)) **by** *auto*
let *?p*=(*var-poss-list* (*lhs* β))!*i*
from *i* **have** *zip* (*var-poss-list* (*lhs* β)) (*map redex-patterns As*) !*i* = (*?p*, *redex-patterns* (*As*!*i*))
using *nth-zip* **by** *auto*
then have *?xs*!*i* = *map* (λ(α, *p*). (α, *?p*@*p*)) (*redex-patterns* (*As*!*i*)) **using** *nth-map i* **by** *auto*
with *j* **obtain** *p*′ **where** *p* = *?p*@*p*′ **and** (α, *p*′) ∈ *set* (*redex-patterns* (*As*!*i*))
by (*smt* (*verit*, *ccfv-threshold*) *case-prod-beta fst-conv imageE list.set-map nth-mem prod.collapse snd-conv*)
with *i* **show** *?thesis* **by** *blast*
qed *simp*
qed

lemma *redex-patterns-elem-fun*′:
assumes (α, *p*) ∈ *set* (*redex-patterns* (*As*!*i*)) **and** *i*:*i* < *length As*
shows (α, *i*#*p*) ∈ *set* (*redex-patterns* (*Pfun f As*))
proof−
let *?xs*=*map2* (λ*i*. *map* (λ(α, *p*). (α, *i* # *p*))) [*0*..<*length As*] (*map redex-patterns As*)
from *i* **have** *zip* [*0*..<*length As*] (*map redex-patterns As*) !*i* = (*i*, *redex-patterns* (*As*!*i*))
using *nth-zip* **by** *auto*
then have *?xs*!*i* = *map* (λ(α, *p*). (α, *i*#*p*)) (*redex-patterns* (*As*!*i*)) **using** *nth-map i* **by** *auto*
with *assms* **have** (α, *i*#*p*) ∈ *set* (*?xs*!*i*) **by** *fastforce*
moreover from *i* **have** *i* < *length ?xs* **by** *simp*
ultimately have ∗:(α, *i*#*p*) ∈ *set* (*concat ?xs*)
unfolding *set-concat* **by** (*meson UN-iff nth-mem*)
then show *?thesis* **by** *simp*
qed

lemma *redex-patterns-elem-rule*′:
assumes (β, *p*) ∈ *set* (*redex-patterns* (*As*!*i*)) **and** *i*:*i* < *length As i* < *length* (*var-poss-list* (*lhs* α))
shows (β, (*var-poss-list* (*lhs* α) ! *i*)@*p*) ∈ *set* (*redex-patterns* (*Prule* α *As*))
proof−
let *?xs*=*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map*

redex-patterns As)
let *?p=var-poss-list* (*lhs* α) ! *i*
from *i* **have** *zip* (*var-poss-list* (*lhs* α)) (*map redex-patterns As*) !*i* = (*?p*, *redex-patterns* (*As*!*i*))
using *nth-zip* **by** *auto*
then have *?xs*!*i* = *map* ($\lambda(\alpha,\ p).\ (\alpha,\ ?p@p)$) (*redex-patterns* (*As*!*i*)) **using** *nth-map i* **by** *auto*
with *assms* **have** (β, *?p*@*p*) ∈ *set* (*?xs*!*i*) **by** *fastforce*
moreover from *i* **have** *i* < *length ?xs* **by** *simp*
ultimately have ∗:(β, *?p*@*p*) ∈ *set* (*concat ?xs*)
unfolding *set-concat* **by** (*meson UN-iff nth-mem*)
then show *?thesis* **by** *simp*
qed

lemma *redex-patterns-elem-subst*:
assumes (α, *p*) ∈ *set* (*redex-patterns* ((*to-pterm t*) · σ))
shows ∃ *p1 p2 x*. *p* = *p1*@*p2* ∧ (α, *p2*) ∈ *set* (*redex-patterns* (σ *x*)) ∧ *p1* ∈ *var-poss t* ∧ *t*|-*p1* = *Var x*
using *assms* **proof**(*induct t arbitrary*: *p*)
case (*Var x*)
then show *?case* **unfolding** *to-pterm.simps eval-term.simps* **by** *force*
next
case (*Fun f ts*)
from *Fun*(*2*) **obtain** *j* **where** *j*:*j* < *length* (*redex-patterns* (*to-pterm* (*Fun f ts*) · σ)) (*redex-patterns* (*to-pterm* (*Fun f ts*) · σ))!*j* = (α, *p*)
by (*metis in-set-idx*)
from *j* **obtain** *i k* **where** *i*:*i* < *length ts*
and *k*:*k* < *length* (*map* ($\lambda(\alpha,\ p).\ (\alpha,\ i\ \#\ p)$) (*redex-patterns* (*to-pterm* (*ts*!*i*) · σ)))
and *rdp*:(*map* ($\lambda(\alpha,\ p).\ (\alpha,\ i\ \#\ p)$) (*redex-patterns* (*to-pterm* (*ts*!*i*) · σ)))!*k* = (α, *p*)
using *nth-concat-split* **unfolding** *length-map to-pterm.simps eval-term.simps redex-patterns.simps* **by** *force*
from *rdp k* **obtain** *p*′ **where** *p*:*p* = *i*#*p*′
by (*smt* (*verit*, *del-insts*) *case-prod-conv list.sel*(*3*) *map-eq-imp-length-eq map-ident nth-map prod.inject surj-pair*)
from *k rdp* **have** (α, *p*′) ∈ *set* (*redex-patterns* (*to-pterm* (*ts*!*i*) · σ))
unfolding *p* **by** (*smt* (*verit*, *del-insts*) *case-prod-conv list.sel*(*3*) *map-eq-imp-length-eq map-ident nth-map nth-mem prod.inject surj-pair*)
with *Fun*(*1*) *i* **obtain** *p1 p2 x* **where** *p*′:*p*′ = *p1*@*p2* **and** *rdp2*:(α, *p2*) ∈ *set* (*redex-patterns* (σ *x*)) **and** *p1* ∈ *var-poss* (*ts*!*i*) **and** (*ts*!*i*)|-*p1* = *Var x*
by (*meson nth-mem*)
with *i* **have** *i*#*p1* ∈ *var-poss* (*Fun f ts*) *Fun f ts* |- (*i*#*p1*) = *Var x*
by *auto*
with *p*′ *rdp2* **show** *?case*
unfolding *p* **by** (*meson Cons-eq-appendI*)
qed

context *left-lin-no-var-lhs*

begin

lemma *redex-patterns-rule''*:
assumes *rdp*:$(\beta, p @ q) \in$ *set* (*redex-patterns* (*Prule* α *As*))
and *wf*:*Prule* α *As* $\in$ *wf-pterm R*
and *p*:$p =$ *var-poss-list* (*lhs* α)!*i*
and *i*:$i <$ *length As*
shows $(\beta, q) \in$ *set* (*redex-patterns* (*As*!*i*))
proof−
from *wf* **obtain** *f ts* **where** *lhs*:*lhs* $\alpha =$ *Fun f ts*
by (*metis Inl-inject case-prodD is-FunE is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *no-var-lhs term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
from *wf i* **have** *l*:*length As* = *length* (*var-poss-list* (*lhs* α))
by (*metis Inl-inject is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *length-var-poss-list length-var-rule term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
with *i p* **have** *p* $\in$ *var-poss* (*Fun f ts*)
by (*metis lhs nth-mem var-poss-list-sound*)
then have $p \neq []$ **by** *force*
then obtain *j p'* **where** *j*:$j <$ *length As* **and** *p'*:$p@q =$ *var-poss-list* (*lhs* α) ! *j* @ p' $(\beta, p') \in$ *set* (*redex-patterns* (*As*!*j*))
using *redex-patterns-elem-rule*[*OF rdp*] **by** *blast*
{assume $j \neq i$
then have $p \perp$ *var-poss-list* (*lhs* α) ! *j*
unfolding *p* **using** *i j* **by** (*metis distinct-var-poss-list l nth-eq-iff-index-eq nth-mem var-poss-list-sound var-poss-parallel*)
with $p'(1)$ **have** *False*
by (*metis less-eq-pos-simps*(*1*) *pos-less-eq-append-not-parallel*)
}
with $p'(1)$ *p* **have** $j = i$ **and** $p' = q$ **by** *fastforce*+
with $p'(2)$ **show** *?thesis* **by** *simp*
qed

lemma *redex-patterns-elem-subst'*:
assumes $(\alpha, p2) \in$ *set* (*redex-patterns* (σ *x*)) **and** *p1*:*p1* $\in$ *poss t t*|-*p1* = *Var x*
shows $(\alpha, p1@p2) \in$ *set* (*redex-patterns* ((*to-pterm t*) $\cdot$ σ))
using *assms* **proof**(*induct t arbitrary*: *p1*)
case (*Var x*)
then show *?case* **unfolding** *to-pterm.simps eval-term.simps* **by** *force*
next
case (*Fun f ts*)
from *Fun*(*3*,*4*) **obtain** *i p1'* **where** *i*:$i <$ *length ts* **and** *p1*:*p1* = *i*#*p1'* **and** *p1'*:*p1'* $\in$ *poss* (*ts*!*i*) (*ts*!*i*)|-*p1'* = *Var x*
by *auto*
with *Fun*(*1*,*2*) **have** $(\alpha, p1' @ p2) \in$ *set* (*redex-patterns* (*to-pterm* (*ts*!*i*) $\cdot$ σ))
using *nth-mem* **by** *blast*
then obtain *j* **where** *j*:$j <$ *length* (*redex-patterns* (*to-pterm* (*ts*!*i*) $\cdot$ σ)) *redex-patterns* (*to-pterm* (*ts*!*i*) $\cdot$ σ)!*j* = $(\alpha, p1' @ p2)$
by (*metis in-set-idx*)
let *?xs*=*map2* ($\lambda i.$ *map* ($\lambda(\alpha, p).$ $(\alpha, i \# p)$)) [*0*..<*length* (*map* ($\lambda s.$ $s \cdot \sigma$) (*map*

to-pterm ts))] (*map redex-patterns* (*map* (λ*s*. *s* · σ) (*map to-pterm ts*)))
from *i j* **have** *rdp*:*?xs*!*i*!*j* = (α, *p1*@*p2*)
unfolding *p1* **by** *auto*
let *?i*=*sum-list* (*map length* (*take i ?xs*)) + *j*
from *rdp i j*(*1*) **have** (*redex-patterns* ((*to-pterm* (*Fun f ts*)) · σ)) ! *?i* = (α, *p1*@*p2*)
using *concat-nth*[*of i ?xs j*] **unfolding** *length-map* **by** *force*
moreover from *i j*(*1*) **have** *?i* < *length* (*redex-patterns* (*to-pterm* (*Fun f ts*) · σ))
using *concat-nth-length*[*of i ?xs j*] **unfolding** *length-map* **by** *force*
ultimately show *?case*
using *nth-mem* **by** *fastforce*
qed

lemma *redex-patterns-join*:
assumes *A* ∈ *wf-pterm R B* ∈ *wf-pterm R A* ⊔ *B* = *Some C*
shows *set* (*redex-patterns C*) = *set* (*redex-patterns A*) ∪ *set* (*redex-patterns B*)
using *assms* **proof**(*induct A arbitrary*: *B C rule*:*subterm-induct*)
case (*subterm A*)
from *subterm*(*2*) **show** *?case* **proof**(*cases A*)
case (*1 x*)
from *subterm*(*2*,*3*,*4*) *var-join* **show** *?thesis*
unfolding *1* **by** *auto*
next
case (*2 As f*)
with *subterm*(*4*) **consider** (*Pfun*) ∃*g Bs*. *B* = *Pfun g Bs* | (*Prule*) ∃α *Bs*. *B* = *Prule* α *Bs* **by** (*meson fun-join*)
then show *?thesis* **proof**(*cases*)
case *Pfun*
then obtain *g Bs* **where** *B*:*B* = *Pfun g Bs* **by** *blast*
from *subterm*(*4*) *join-fun-fun* **obtain** *Cs* **where** *fg*:*f* = *g* **and** *l-As-Bs*:*length As* = *length Bs* **and**
C:*C* = *Pfun f Cs* **and** *l-Cs-As*:*length Cs* = *length As* **and** *Cs*:(∀ *i*<*length As*. *As* ! *i* ⊔ *Bs* ! *i* = *Some* (*Cs* ! *i*))
unfolding *2 B* **by** *force*
{fix *i* **assume** *i*:*i* < *length As*
with *subterm*(*3*) **have** *Bs*!*i* ∈ *wf-pterm R*
using *B l-As-Bs* **by** *auto*
with *subterm*(*1*) *i 2 Cs* **have** *set* (*redex-patterns* (*Cs*!*i*)) = *set* (*redex-patterns* (*As*!*i*)) ∪ *set* (*redex-patterns* (*Bs*!*i*))
by (*meson nth-mem supt.arg*)
}note *IH*=*this*
{fix α *p* **assume** (α, *p*) ∈ *set* (*redex-patterns C*)
then obtain *i p*′ **where** *i*:*i* < *length Cs* **and** *p*:*p* = *i*#*p*′ **and** (α, *p*′) ∈ *set* (*redex-patterns* (*Cs*!*i*))
unfolding *C* **by** (*meson redex-patterns-elem-fun*)
with *IH* **consider** (α, *p*′) ∈ *set* (*redex-patterns* (*As*!*i*)) | (α, *p*′) ∈ *set* (*redex-patterns* (*Bs*!*i*))
using *l-Cs-As* **by** *fastforce*

```
        then have (α, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
proof(cases)
       case 1
       with i have (α, p) ∈ set (redex-patterns A)
        unfolding 2 p l-Cs-As by (meson redex-patterns-elem-fun')
       then show ?thesis by simp
      next
       case 2
       with i have (α, p) ∈ set (redex-patterns B)
        unfolding B l-Cs-As l-As-Bs p by (meson redex-patterns-elem-fun')
       then show ?thesis by simp
      qed
    }
    moreover
    {fix α p assume (α, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
     then consider (α, p) ∈ set (redex-patterns A) | (α, p) ∈ set (redex-patterns
B) by force
      then have (α, p) ∈ set (redex-patterns C) proof(cases)
       case 1
        then obtain i p' where i:i < length As and p:p = i#p' and (α, p') ∈
set (redex-patterns (As!i))
         unfolding 2 by (meson redex-patterns-elem-fun)
       with IH have (α, p') ∈ set (redex-patterns (Cs!i)) by blast
       with i l-Cs-As show ?thesis unfolding C p
        by (metis redex-patterns-elem-fun')
      next
       case 2
        then obtain i p' where i:i < length Bs and p:p = i#p' and (α, p') ∈
set (redex-patterns (Bs!i))
         unfolding B by (meson redex-patterns-elem-fun)
       with IH l-As-Bs have (α, p') ∈ set (redex-patterns (Cs!i)) by simp
       with i l-Cs-As l-As-Bs show ?thesis unfolding C p
        by (metis redex-patterns-elem-fun')
      qed
    }
    ultimately show ?thesis by auto
  next
    case Prule
    then obtain α Bs where B:B = Prule α Bs by blast
    from B subterm(3) have alpha:to-rule α ∈ R
     using wf-pterm.simps by fastforce
    then obtain f' ts where lhs:lhs α = Fun f' ts
     using no-var-lhs by fastforce
    from alpha have lin:linear-term (lhs α)
     using left-lin left-linear-trs-def by fastforce
   from B subterm(3,4) obtain σ Cs where sigma:match A (to-pterm (lhs α))
= Some σ
    and C:C = Prule α Cs and l-Cs-Bs:length Cs = length Bs and Cs:(∀ i<length
Bs. σ (var-rule α ! i) ⊔ (Bs ! i) = Some (Cs ! i))
```

unfolding *2* **using** *join-rule-fun join-sym* **by** (*smt* (*verit*, *best*))
from *B subterm*(*3*) **have** *l-Bs*:*length Bs* = *length* (*var-rule* α)
using *wf-pterm.simps* **by** *fastforce*
from *sigma* **have** *A*:*A* = (*to-pterm* (*lhs* α) $\cdot$ σ)
by (*simp add*: *match-matches*)
{fix *i* **assume** *i*:*i* < *length Bs*
with *sigma lhs l-Bs* **have** σ (*var-rule* α ! *i*) $\lhd$ *A*
by (*smt* (*verit*, *best*) *comp-def match-lhs-subst nth-mem set-remdups set-rev set-vars-term-list subst-image-subterm to-pterm.simps*(*2*) *vars-to-pterm*)
moreover have σ (*var-rule* α ! *i*) $\in$ *wf-pterm R*
using *subterm*(*2*) **by** (*metis i l-Bs match-well-def sigma vars-to-pterm*)
moreover from *i subterm*(*3*) **have** *Bs*!*i* $\in$ *wf-pterm R*
using *B nth-mem* **by** *blast*
ultimately have *set* (*redex-patterns* (*Cs*!*i*)) = *set* (*redex-patterns* (σ (*var-rule* α ! *i*))) $\cup$ *set* (*redex-patterns* (*Bs*!*i*))
using *subterm*(*1*) *Cs l-Cs-Bs i* **by** *presburger*
}note *IH*=*this*
{fix β *p* **assume** *rdp*:(β, *p*) $\in$ *set* (*redex-patterns C*)
then have (β, *p*) $\in$ *set* (*redex-patterns A*) $\cup$ *set* (*redex-patterns B*) **proof**(*cases p*=[])
case *True*
with *rdp* **have** α = β
unfolding *C* **using** *lhs* **by** (*metis* (*no-types*, *lifting*) *C join-wf-pterm list.set-intros*(*1*) *option.sel*
prod.inject redex-patterns.simps(*3*) *redex-patterns-label subterm.prems*(*1*) *subterm.prems*(*2*) *subterm.prems*(*3*))
then show *?thesis* **unfolding** *B redex-patterns.simps True* **by** *simp*
next
case *False*
with *rdp* **obtain** *i p'* **where** *i*:*i* < *length Cs i* < *length* (*var-poss-list* (*lhs* α))
and *p*:*p* = (*var-poss-list* (*lhs* α) ! *i*)@*p'* **and** $*$:(β, *p'*) $\in$ *set* (*redex-patterns* (*Cs*!*i*))
unfolding *C* **by** (*meson redex-patterns-elem-rule*)
let *?p*=*var-poss-list* (*lhs* α) ! *i*
from $*$ *i IH* **consider** (β, *p'*) $\in$ *set* (*redex-patterns* (σ (*var-rule* α ! *i*))) | (β, *p'*) $\in$ *set* (*redex-patterns* (*Bs*!*i*))
using *l-Cs-Bs* **by** *fastforce*
then show *?thesis* **proof**(*cases*)
case *1*
let *?x*=*var-rule* α ! *i*
from *i*(*2*) **have** *p-pos*:*?p* $\in$ *poss* (*lhs* α)
by (*metis nth-mem var-poss-iff var-poss-list-sound*)
from *i*(*2*) **have** *p-x*:(*lhs* α)|-*?p* = *Var ?x*
by (*metis* ‹*to-rule* α $\in$ *R*› *case-prodD left-lin left-linear-trs-def length-var-poss-list linear-term-var-vars-term-list vars-term-list-var-poss-list*)
from *i*(*2*) **have** (β, *p*) $\in$ *set* (*redex-patterns A*)
unfolding *p A* **using** *redex-patterns-elem-subst'*[*of* β *p'* σ *?x*, *OF 1 p-pos p-x*] **by** *simp*

```
        then show ?thesis by simp
      next
        case 2
        from i have (β, p) ∈ set (redex-patterns B)
            unfolding B p l-Cs-Bs using redex-patterns-elem-rule'[OF 2] by
presburger
        then show ?thesis by simp
    qed
    qed
  }
  moreover
  {fix β p assume (β, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
   then consider (β, p) ∈ set (redex-patterns A) | (β, p) ∈ set (redex-patterns
B) by force
    then have (β, p) ∈ set (redex-patterns C) proof(cases)
     case 1
       then obtain p1 p2 x where p:p = p1@p2 and rdp2:(β, p2) ∈ set
(redex-patterns (σ x))
        and p1:p1 ∈ var-poss (lhs α) lhs α|-p1 = Var x
        unfolding A using redex-patterns-elem-subst by metis
      then obtain i where i:i < length (var-rule α) (var-rule α)!i = x
     using lin by (metis in-set-conv-nth length-var-poss-list linear-term-var-vars-term-list
term.inject(1) var-poss-list-sound vars-term-list-var-poss-list)
      with p1 lin have p1:p1 = var-poss-list (lhs α) ! i
     by (metis length-var-poss-list linear-term-unique-vars linear-term-var-vars-term-list
nth-mem var-poss-imp-poss var-poss-list-sound vars-term-list-var-poss-list)
      from i IH rdp2 have (β, p2) ∈ set (redex-patterns (Cs!i))
        by (simp add: l-Bs)
      with i(1) show ?thesis unfolding C p
           using redex-patterns-elem-rule' p1 by (metis alpha l-Bs l-Cs-Bs
length-var-poss-list length-var-rule)
    next
      case 2
      show ?thesis proof(cases p=[])
        case True
        from 2 have α = β
     unfolding B True using lhs by (metis (no-types, lifting) B list.set-intros(1)
option.sel
       prod.inject redex-patterns.simps(3) redex-patterns-label subterm.prems(2))

        then show ?thesis unfolding C redex-patterns.simps True by simp
      next
        case False
        with 2 obtain i p' where i:i < length Bs i < length (var-poss-list (lhs
α))
       and p:p = (var-poss-list (lhs α) ! i)@p' and ∗:(β, p') ∈ set (redex-patterns
(Bs!i))
          unfolding B by (meson redex-patterns-elem-rule)
        with IH l-Cs-Bs have (β, p') ∈ set (redex-patterns (Cs!i)) by simp
```

```
        with i l-Cs-Bs show ?thesis unfolding C p
          by (metis redex-patterns-elem-rule')
       qed
      qed
     }
     ultimately show ?thesis by auto
    qed
  next
    case (3 α As)
    from 3(2) obtain f' ts where lhs:lhs α = Fun f' ts
     using no-var-lhs by fastforce
    from 3(2) have lin:linear-term (lhs α)
     using left-lin left-linear-trs-def by fastforce
   from 3 subterm(2,4) consider (Pfun) ∃ g Bs. B = Pfun g Bs | (Prule) ∃ β Bs.
B = Prule β Bs by (meson rule-join)
    then show ?thesis proof(cases)
     case Pfun
     then obtain f Bs where B:B = Pfun f Bs by blast
     from subterm(2,4) obtain σ Cs where sigma:match B (to-pterm (lhs α)) =
Some σ
     and C:C = Prule α Cs and l-Cs-As:length Cs = length As and As:(∀ i<length
As. (As ! i) ⊔ σ (var-rule α ! i) = Some (Cs ! i))
       unfolding 3 B using 3(3) join-rule-fun by metis
     {fix i assume i:i < length As
      have σ (var-rule α ! i) ∈ wf-pterm R
       using subterm(3) by (metis 3(3) i match-well-def sigma vars-to-pterm)
      then have set (redex-patterns (Cs!i)) = set (redex-patterns (As!i)) ∪ set
(redex-patterns (σ (var-rule α ! i)))
       using subterm(1) 3 by (meson As i nth-mem supt.arg)
     }note IH=this
     {fix β p assume rdp:(β, p) ∈ set (redex-patterns C)
        then have (β, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
proof(cases p=[])
       case True
       with rdp have α = β
         unfolding C using lhs by (metis (no-types, lifting) C join-wf-pterm
list.set-intros(1) option.sel
        prod.inject redex-patterns.simps(3) redex-patterns-label subterm.prems(1)
subterm.prems(2) subterm.prems(3))
       then show ?thesis unfolding 3 redex-patterns.simps True by simp
      next
       case False
       with rdp obtain i p' where i:i < length Cs i < length (var-poss-list (lhs
α))
       and p:p = (var-poss-list (lhs α) ! i)@p' and ∗:(β, p') ∈ set (redex-patterns
(Cs!i))
        unfolding C by (meson redex-patterns-elem-rule)
       let ?p=var-poss-list (lhs α) ! i
       from ∗ i IH consider (β, p') ∈ set (redex-patterns (σ (var-rule α ! i))) |
```

```
(β, p') ∈ set (redex-patterns (As!i))
          using l-Cs-As by auto
        then show ?thesis proof(cases)
          case 1
          let ?x=var-rule α ! i
          from i(2) have p-pos:?p ∈ poss (lhs α)
            by (metis nth-mem var-poss-iff var-poss-list-sound)
          from i(2) have p-x:(lhs α)|-?p = Var ?x
           by (metis 3(2) case-prodD left-lin left-linear-trs-def length-var-poss-list
linear-term-var-vars-term-list vars-term-list-var-poss-list)
          from sigma have (β, p) ∈ set (redex-patterns B)
           unfolding p using redex-patterns-elem-subst'[of β p' σ ?x, OF 1 p-pos
p-x] by (simp add: match-matches)
          then show ?thesis by simp
        next
          case 2
          from i have (β, p) ∈ set (redex-patterns A)
             unfolding 3(1) p l-Cs-As using redex-patterns-elem-rule'[OF 2] by
presburger
          then show ?thesis by simp
        qed
      qed
    }
    moreover
    {fix β p assume (β, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
     then consider (β, p) ∈ set (redex-patterns B) | (β, p) ∈ set (redex-patterns
A)  by force
      then have (β, p) ∈ set (redex-patterns C) proof(cases)
        case 1
          then obtain p1 p2 x where p:p = p1@p2 and rdp2:(β, p2) ∈ set
(redex-patterns (σ x))
          and p1:p1 ∈ var-poss (lhs α) lhs α|-p1 = Var x
          using sigma redex-patterns-elem-subst using match-matches by blast
        then obtain i where i:i < length (var-rule α) (var-rule α)!i = x
       using lin by (metis in-set-conv-nth length-var-poss-list linear-term-var-vars-term-list
term.inject(1) var-poss-list-sound vars-term-list-var-poss-list)
        with p1 lin have p1:p1 = var-poss-list (lhs α) ! i
       by (metis length-var-poss-list linear-term-unique-vars linear-term-var-vars-term-list
nth-mem var-poss-imp-poss var-poss-list-sound vars-term-list-var-poss-list)
        from i IH rdp2 have (β, p2) ∈ set (redex-patterns (Cs!i))
          by (simp add: 3(3))
        with i(1) show ?thesis unfolding C p
              using redex-patterns-elem-rule' p1 by (metis 3(2) 3(3) l-Cs-As
length-var-poss-list length-var-rule)
      next
        case 2
        show ?thesis proof(cases p=[])
          case True
          from 2 have α = β
```

unfolding *True* **using** *lhs* **by** (*metis 3(1) list.set-intros(1) option.sel prod.sel(1) redex-patterns.simps(3) redex-patterns-label subterm.prems(1)*)
then show *?thesis* **unfolding** *C redex-patterns.simps True* **by** *simp*
next
case *False*
with *2* **obtain** *i p′* **where** *i*:*i < length As i < length* (*var-poss-list* (*lhs α*))
and *p*:*p* = (*var-poss-list* (*lhs α*) ! *i*)@*p′* **and** ∗:(*β*, *p′*) ∈ *set* (*redex-patterns* (*As*!*i*))
using *3(1) redex-patterns-elem-rule* **by** *blast*
with *IH l-Cs-As* **have** (*β*, *p′*) ∈ *set* (*redex-patterns* (*Cs*!*i*)) **by** *simp*
with *i l-Cs-As* **show** *?thesis* **unfolding** *C p*
by (*metis redex-patterns-elem-rule′*)
qed
qed
}
ultimately show *?thesis* **by** *auto*
next
case *Prule*
then obtain *β Bs* **where** *B*:*B* = *Prule β Bs* **by** *blast*
obtain *Cs* **where** *alpha-beta*:*α* = *β* **and** *l-As-Bs*:*length As* = *length Bs*
and *C*:*C* = *Prule α Cs* **and** *l-Cs-As*:*length Cs* = *length As* **and** *args*:∀ *i* < *length As*. *As* ! *i* ⊔ *Bs* ! *i* = *Some* (*Cs* ! *i*)
using *join-rule-rule*[*OF subterm*(*4*,*2*,*3*)[*unfolded B 3*]] **using** *3(3)* **by** *fastforce*
{fix *i* **assume** *i < length As*
from *subterm(1)* **have** *set* (*redex-patterns* (*Cs*!*i*)) = *set* (*redex-patterns* (*As*!*i*)) ∪ *set* (*redex-patterns* (*Bs*!*i*))
by (*metis 3(1) 3(4) B* ‹*i < length As*› *fun-well-arg l-As-Bs local.args nth-mem subterm.prems(2) supt.arg*)
}note *IH*=*this*
{fix *γ p* **assume** *rdp*:(*γ*, *p*) ∈ *set* (*redex-patterns C*)
have (*γ*, *p*) ∈ *set* (*redex-patterns A*) ∪ *set* (*redex-patterns B*) **proof**(*cases p* = [])
case *True*
from *rdp* **have** *α* = *γ* **unfolding** *C True* **using** *lhs*
by (*metis* (*no-types*, *lifting*) *C join-wf-pterm list.set-intros(1) option.sel prod.inject redex-patterns.simps(3) redex-patterns-label subterm*(*2*,*3*,*4*))
then show *?thesis* **unfolding** *3 True* **by** *simp*
next
case *False*
then obtain *p2 i* **where** *i*:*i < length Cs i < length* (*var-poss-list* (*lhs α*))
and *p*:*p* = *var-poss-list* (*lhs α*) ! *i* @*p2* **and** (*γ*, *p2*) ∈ *set* (*redex-patterns* (*Cs*!*i*))
using *C rdp redex-patterns-elem-rule* **by** *blast*
with *IH* **consider** (*γ*, *p2*) ∈ *set* (*redex-patterns* (*As*!*i*)) | (*γ*, *p2*) ∈ *set* (*redex-patterns* (*Bs*!*i*))
using *l-Cs-As* **by** *fastforce*
then show *?thesis* **proof**(*cases*)

```
          case 1
          with i have (γ, p) ∈ set (redex-patterns A)
            unfolding 3 p l-Cs-As by (metis 3(3) redex-patterns-elem-rule')
          then show ?thesis by simp
        next
          case 2
          with i have (γ, p) ∈ set (redex-patterns B)
         unfolding B l-Cs-As l-As-Bs p alpha-beta using redex-patterns-elem-rule'
by blast
          then show ?thesis by simp
        qed
      qed
    }
    moreover
    {fix γ p assume rdp:(γ, p) ∈ set (redex-patterns A) ∪ set (redex-patterns B)
      have (γ, p) ∈ set (redex-patterns C) proof(cases p = [])
        case True
        from rdp lhs have γ = α
          unfolding 3 B alpha-beta True
       by (metis 3(1) B Un-iff alpha-beta list.set-intros(1) option.inject prod.inject
redex-patterns.simps(3) redex-patterns-label subterm.prems(1) subterm.prems(2))
        then show ?thesis unfolding C True by simp
      next
        case False
            from rdp consider (γ, p) ∈ set (redex-patterns A) | (γ, p) ∈ set
(redex-patterns B) by force
        then show ?thesis proof(cases)
          case 1
           then obtain p2 i where i:i < length As i < length (var-poss-list (lhs
α))
          and p:p = var-poss-list (lhs α) ! i @p2 and (γ, p2) ∈ set (redex-patterns
(As!i))
            using 3 redex-patterns-elem-rule False by blast
          with IH have (γ, p2) ∈ set (redex-patterns (Cs!i)) by blast
          with i l-Cs-As show ?thesis unfolding C p
            by (metis redex-patterns-elem-rule')
        next
          case 2
           then obtain p2 i where i:i < length Bs i < length (var-poss-list (lhs
α))
          and p:p = var-poss-list (lhs α) ! i @p2 and (γ, p2) ∈ set (redex-patterns
(Bs!i))
            using B alpha-beta redex-patterns-elem-rule False by blast
           with IH have (γ, p2) ∈ set (redex-patterns (Cs!i)) using l-As-Bs by
simp
          with i l-Cs-As l-As-Bs show ?thesis unfolding C p
            by (metis redex-patterns-elem-rule')
        qed
      qed
```

```
    }
    ultimately show ?thesis by auto
   qed
  qed
qed

lemma redex-patterns-join-list:
  assumes join-list As = Some A and ∀ a ∈ set As. a ∈ wf-pterm R
  shows set (redex-patterns A) = ⋃ (set (map (set ∘ redex-patterns) As))
  using assms proof(induct As arbitrary:A)
  case (Cons a As)
  show ?case proof(cases As = [])
    case True
    from Cons(2,3) have a = A
      unfolding True join-list.simps by (simp add: join-with-source)
    then show ?thesis unfolding True by simp
  next
    case False
    then have ∗:join-list (a#As) = join-opt a (join-list As)
      using join-list.elims by blast
    with Cons(2) obtain A' where A':join-list As = Some A' by fastforce
      with Cons(1,3) have set (redex-patterns A') = ⋃ (set (map (set ∘ re-
dex-patterns) As))
      by simp
   then show ?thesis using redex-patterns-join ∗ Cons(2,3) unfolding A' join-opt.simps
      by (metis (no-types, opaque-lifting) A' Sup-insert insert-iff join-list-wf-pterm
list.set(2) list.simps(9) o-apply)
  qed
qed simp

lemma redex-patterns-context:
  assumes p ∈ poss s
   shows redex-patterns ((ctxt-of-pos-term p (to-pterm s)) ⟨A⟩) = map (λ(α, q).
(α,p@q)) (redex-patterns A)
  using assms proof(induct p arbitrary:s)
  case (Cons i p')
  from Cons(2) obtain f ss where s:s = Fun f ss
    by (meson args-poss)
  from Cons(2) have i:i < length ss and p':p' ∈ poss (ss!i)
    unfolding s by auto
  with Cons(1) have IH:redex-patterns (ctxt-of-pos-term p' (to-pterm (ss!i)))⟨A⟩
=
    map (λ(α, q). (α,p'@q)) (redex-patterns A) by simp
   from i have l:length (take i (map to-pterm ss) @ (ctxt-of-pos-term p' (map
to-pterm ss ! i))⟨A⟩ # drop (Suc i) (map to-pterm ss)) = length ss
    by simp
  let ?take-i=take i (map to-pterm ss)
  let ?ith=(ctxt-of-pos-term p' (map to-pterm ss ! i))⟨A⟩
  let ?drop-i=drop (Suc i) (map to-pterm ss)
```

let *?xs=take i* (*map to-pterm ss*) @ (*ctxt-of-pos-term p'* (*map to-pterm ss* ! *i*))⟨*A*⟩ # *drop* (*Suc i*) (*map to-pterm ss*)
let *?zip=zip* [*0..<length ss*] (*map redex-patterns ?xs*)
from *i* **have** *l-zip*:*length ?zip* = *length ss* **by** *auto*
let *?zip1=zip* [*0..<i*] (*map redex-patterns ?take-i*)
let *?zip2=zip* [*Suc i..<length ss*] (*map redex-patterns ?drop-i*)
have *zip*:*?zip* = *?zip1* @ ((*i*, *redex-patterns ?ith*) # *?zip2*)
unfolding *map-append zip-append2* **using** *i* **by** (*simp add*: *upt-conv-Cons*)
{fix *j* **assume** *j*:*j* < *length* (*map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip1*)
with *i* **have** (*map redex-patterns ?take-i*)!*j* = []
by (*simp add*: *redex-patterns-to-pterm*)
with *j* **have** *?zip1* ! *j* = (*j*, [])
by *simp*
with *j* **have** *map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip1* ! *j* = []
by *simp*
}
then have *1*:*concat* (*map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip1*) = []
by (*metis length-0-conv length-greater-0-conv less-nat-zero-code nth-concat-split*)

{fix *j* **assume** *j*:*j* < *length* (*map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip2*)
with *i* **have** (*map redex-patterns ?drop-i*)!*j* = []
by (*simp add*: *redex-patterns-to-pterm*)
with *j* **have** *?zip2* ! *j* = (*j*+*Suc i*, [])
by *simp*
with *j* **have** *map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip2* ! *j* = []
by *simp*
}
then have *2*:*concat* (*map* (λ(*x*, *y*). *map* (λ(α, *p*). (α, *x* # *p*)) *y*) *?zip2*) = []
by (*metis length-0-conv length-greater-0-conv less-nat-zero-code nth-concat-split*)

show *?case* **unfolding** *s to-pterm.simps ctxt-of-pos-term.simps intp-actxt.simps redex-patterns.simps l zip*
unfolding *map-append concat-append 1 list.map*(*2*) *concat.simps 2* **using** *IH i* **by** *simp*
qed *simp*

lemma *redex-patterns-prule*:
assumes *l*:*length ts* = *length* (*var-poss-list* (*lhs* α))
shows *redex-patterns* (*Prule* α (*map to-pterm ts*)) = [(α, [])]
proof−
{fix *x* **assume** *x*:*x* ∈ *set* (*map2* (λ*x*. *map* (λ(α, *p2*). (α, *x* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map redex-patterns* (*map to-pterm ts*)))
from *l* **have** *length* (*map2* (λ*x*. *map* (λ(α, *p2*). (α, *x* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map redex-patterns* (*map to-pterm ts*))) = *length* (*var-poss-list* (*lhs* α))
by *simp*
with *x* **obtain** *i* **where** *i*:*i* < *length* (*var-poss-list* (*lhs* α)) *x* = (*map2* (λ*x*. *map* (λ(α, *p2*). (α, *x* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map redex-patterns* (*map to-pterm ts*)))!*i*
by (*metis in-set-idx*)

```
    from i l have x = []
      using redex-patterns-to-pterm by simp
  }
  then show ?thesis
    unfolding redex-patterns.simps using concat-eq-Nil-conv by blast
qed

lemma redex-patterns-single:
  assumes p ∈ poss s and to-rule α ∈ R
  shows redex-patterns (ll-single-redex s p α) = [(α, p)]
proof−
  let ?As=map (to-pterm ∘ (λpi. s |- (p @ pi))) (var-poss-list (lhs α))
  let ?A=Prule α ?As
  have redex-patterns ?A = [(α, [])]
    using redex-patterns-prule using length-map by fastforce
  moreover have set (redex-patterns (ll-single-redex s p α)) = set (map (λ (α, q).
(α,p@q)) (redex-patterns ?A))
     using redex-patterns-context assms redex-patterns-to-pterm[of s] unfolding
ll-single-redex-def using p-in-poss-to-pterm by fastforce
  ultimately have set:set (redex-patterns (ll-single-redex s p α)) = {(α, p)}
    by simp
  have wf:ll-single-redex s p α ∈ wf-pterm R
    using assms left-lin left-linear-trs-def single-redex-wf-pterm by fastforce
  have sorted:sorted-wrt (ord.lexordp (<)) (map snd [(α, p)]) by simp
  show ?thesis using redex-patterns-equal[OF wf sorted] set by simp
qed

lemma get-label-imp-rdp:
  assumes get-label (labeled-source A |- p) = Some (α, 0)
    and A ∈ wf-pterm R
    and p ∈ poss (labeled-source A)
  shows (α, p) ∈ set (redex-patterns A)
  using assms proof(induct A arbitrary:p)
  case (Pfun f As)
  then show ?case proof(cases p)
    case (Cons i p′)
    from Pfun(4) have i:i < length As
      unfolding Cons by simp
    moreover from Pfun(2,4) have get-label (labeled-source (As!i) |- p′) = Some
(α, 0)
      unfolding Cons by simp
    moreover from Pfun(4) have p′ ∈ poss (labeled-source (As!i))
      unfolding Cons using i by simp
    ultimately have (α, p′) ∈ set (redex-patterns (As!i))
      using Pfun(1,3) using nth-mem by blast
    then show ?thesis
     unfolding Cons redex-patterns.simps using i by (metis redex-patterns.simps(2)
redex-patterns-elem-fun′)
  qed simp
```

next
case (*Prule* β *As*)
from *Prule*(*3*) **obtain** *f ts* **where** *lhs*:*lhs* β = *Fun f ts*
by (*metis Inl-inject Term.term.simps*(*4*) *case-prodD is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *no-var-lhs term.collapse*(*2*) *term.sel*(*2*) *wf-pterm.simps*)
then show *?case* **proof**(*cases p*)
case *Nil*
from *Prule*(*2*,*3*,*4*) **show** *?thesis*
unfolding *Nil labeled-source.simps lhs label-term.simps eval-term.simps subt-at.simps get-label.simps* **by** *simp*
next
case (*Cons i' p'*)
from *Prule*(*3*) **have** *l*:*length As* = *length* (*var-poss-list* (*lhs* β))
by (*metis Inl-inject is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *length-var-poss-list length-var-rule term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
from *Prule* **obtain** *i p2* **where** *p*:*p* = *var-poss-list* (*lhs* β)!*i* @ *p2* **and** *i*:*i* < *length As* **and** *p2*:*p2* $\in$ *poss* (*labeled-source* (*As*!*i*))
by (*smt* (*verit*) *labeled-source-to-term left-lin-no-var-lhs.get-label-Prule left-lin-no-var-lhs-axioms list.distinct*(*1*) *local.Cons poss-term-lab-to-term*)
let *?x*=*vars-term-list* (*lhs* β) ! *i*
let *?p1*=*var-poss-list* (*lhs* β) ! *i*
have *p1*:*?p1* $\in$ *poss* (*labeled-lhs* β)
by (*metis i l label-term-to-term nth-mem poss-term-lab-to-term var-poss-imp-poss var-poss-list-sound*)
have *labeled-lhs* β |- *?p1* = *Var ?x*
using *i l* **by** (*metis length-var-poss-list var-poss-list-labeled-lhs vars-term-list-labeled-lhs vars-term-list-var-poss-list*)
then have *labeled-source* (*Prule* β *As*) |- *?p1* = *labeled-source* (*As*!*i*)
unfolding *labeled-source.simps subt-at-subst*[*OF p1*]
by (*smt* (*verit*) *Inl-inject Prule.prems*(*2*) *apply-lhs-subst-var-rule comp-eq-dest-lhs eval-term.simps*(*1*) *i is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *l length-map length-remdups-eq length-rev length-var-poss-list map-nth-conv rev-rev-ident term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)
with *Prule*(*2*,*4*) **have** *get-label* (*labeled-source* (*As*!*i*)|-*p2*) = *Some* (α, *0*)
unfolding *p labeled-source.simps* **by** *auto*
then have (α, *p2*) $\in$ *set* (*redex-patterns* (*As*!*i*))
using *Prule*(*1*)[*of As*!*i p2*] *p2 Prule*(*3*) *i* **by** (*meson fun-well-arg nth-mem*)
then show *?thesis* **unfolding** *p redex-patterns.simps* **using** *i*
by (*metis l redex-patterns.simps*(*3*) *redex-patterns-elem-rule'*)
qed
qed *simp*

lemma *redex-pattern-proof-term-equality*:
assumes *A* $\in$ *wf-pterm R B* $\in$ *wf-pterm R*
and *set* (*redex-patterns A*) = *set* (*redex-patterns B*)
and *source A* = *source B*
shows *A* = *B*
using *assms* **proof**(*induct A arbitrary*:*B*)
case (*1 x*)

```
  then show ?case
    using redex-poss-empty-imp-empty-step source-empty-step by force
next
  case (2 As f)
  then show ?case proof(cases B)
    case (Pfun g Bs)
    from 2(4) have f:f = g
      unfolding Pfun by fastforce
    from 2(4) have len:length As = length Bs
      unfolding Pfun f by (metis length-map source.simps(2) term.inject(2))
    {fix i assume i:i < length As
       have set (redex-patterns (As!i)) = set (redex-patterns (Bs!i)) proof(rule
ccontr)
        assume set (redex-patterns (As!i)) ≠ set (redex-patterns (Bs!i))
       then consider ∃ r. r ∈ set (redex-patterns (As!i)) ∧ r ∉ set (redex-patterns
(Bs!i)) |
                     ∃ r. r ∈ set (redex-patterns (Bs!i)) ∧ r ∉ set (redex-patterns
(As!i))
         by blast
       then show False proof(cases)
         case 1
        then obtain p α where (α, p) ∈ set (redex-patterns (As!i)) and B:(α, p)
∉ set (redex-patterns (Bs!i))
           by force
         then have (α, i#p) ∈ set (redex-patterns (Pfun f As))
           by (meson i redex-patterns-elem-fun')
         moreover from B have (α, i#p) ∉ set (redex-patterns (Pfun f Bs))
           by (metis list.inject redex-patterns-elem-fun)
         ultimately show ?thesis
           using 2.prems(2) Pfun f by blast
       next
         case 2
          then obtain p α where (α, p) ∈ set (redex-patterns (Bs!i)) and A:(α,
p) ∉ set (redex-patterns (As!i))
           by force
         then have (α, i#p) ∈ set (redex-patterns (Pfun f Bs))
           by (metis i len redex-patterns-elem-fun')
         moreover from A have (α, i#p) ∉ set (redex-patterns (Pfun f As))
           by (metis list.inject redex-patterns-elem-fun)
         ultimately show ?thesis
           using 2.prems(2) Pfun f by blast
       qed
     qed
     moreover have (Bs!i) ∈ wf-pterm R
       using 2(2) Pfun i len by auto
     ultimately have As!i = Bs!i
         using 2(1,4) by (metis Pfun i len nth-map nth-mem source.simps(2)
term.inject(2))
   }
```

```
    then show ?thesis
      unfolding Pfun f using len using nth-equalityI by blast
  next
    case (Prule α Bs)
    with 2(3) show ?thesis
    by (metis list.distinct(1) list.set-intros(1) redex-patterns.simps(3) redex-patterns-elem-fun)

  qed simp
next
  case (3 α As)
  then show ?case proof(cases B)
    case (Pfun g Bs)
    with 3(5) show ?thesis
    by (metis list.distinct(1) list.set-intros(1) redex-patterns.simps(3) redex-patterns-elem-fun)
  next
    case (Prule β Bs)
    from 3(5) have α:α = β
      unfolding Prule using distinct-snd-rdp
     by (metis 3.prems(1) Pair-inject Prule left-lin-no-var-lhs.redex-patterns-label
       left-lin-no-var-lhs-axioms list.set-intros(1) option.inject redex-patterns.simps(3))
    from 3 have len:length As = length Bs
      using Prule α by (metis length-args-well-Prule wf-pterm.intros(3))
    have len2:length (var-poss-list (lhs β)) = length Bs
      by (metis 3.hyps(1) 3.hyps(2) α len length-var-poss-list length-var-rule)
    {fix i assume i:i < length As
      obtain pi where pi:var-poss-list (lhs β) ! i = pi
        by auto
        have set (redex-patterns (As!i)) = set (redex-patterns (Bs!i)) proof(rule
ccontr)
        assume set (redex-patterns (As!i)) ≠ set (redex-patterns (Bs!i))
       then consider ∃r. r ∈ set (redex-patterns (As!i)) ∧ r ∉ set (redex-patterns
(Bs!i)) |
                     ∃r. r ∈ set (redex-patterns (Bs!i)) ∧ r ∉ set (redex-patterns
(As!i))
         by blast
       then show False proof(cases)
         case 1
        then obtain p β where (β, p) ∈ set (redex-patterns (As!i)) and B:(β, p)
∉ set (redex-patterns (Bs!i))
           by force
         then show False
             using 3(4,5) by (metis Prule α i len len2 redex-patterns-elem-rule'
redex-patterns-rule'')
       next
         case 2
          then obtain p β where (β, p) ∈ set (redex-patterns (Bs!i)) and A:(β,
p) ∉ set (redex-patterns (As!i))
           by force
         then show False
```

using *3* **by** (*metis Prule* α *i len len2 redex-patterns-elem-rule′ redex-patterns-rule′′ wf-pterm.intros*(*3*))
qed
qed
moreover have (*Bs*!*i*) ∈ *wf-pterm R*
using *3.prems*(*1*) *Prule i len* **by** *auto*
moreover have *co-initial* (*As*!*i*) (*Bs*!*i*)
using *3* **by** (*metis Prule* α *co-init-prule i wf-pterm.intros*(*3*))
ultimately have *As*!*i* = *Bs*!*i*
using *3*(*3*) **by** (*simp add*: *i*)
}
then show *?thesis*
unfolding *Prule* α **using** *len* **using** *nth-equalityI* **by** *blast*
qed *simp*
qed

end

abbreviation *single-steps* :: (′*f*, ′*v*) *pterm* ⇒ (′*f*, ′*v*) *pterm list*
where *single-steps A* ≡ *map* (λ (α, *p*). *ll-single-redex* (*source A*) *p* α) (*redex-patterns A*)

context *left-lin-wf-trs*
begin

lemma *ll-no-var-lhs*: *left-lin-no-var-lhs R*
by (*simp add*: *left-lin-axioms left-lin-no-var-lhs-def no-var-lhs-axioms*)

lemma *single-step-redex-patterns*:
assumes *A* ∈ *wf-pterm R* Δ ∈ *set* (*single-steps A*)
shows ∃ *p* α. Δ = *ll-single-redex* (*source A*) *p* α ∧ (α, *p*) ∈ *set* (*redex-patterns A*) ∧ *redex-patterns* Δ = [(α, *p*)]
proof−
from *assms* **obtain** *p* α **where** Δ:Δ = *ll-single-redex* (*source A*) *p* α **and** *rdp*:(α, *p*) ∈ *set* (*redex-patterns A*)
by *auto*
moreover have *to-rule* α ∈ *R*
using *rdp assms*(*1*) *labeled-wf-pterm-rule-in-TRS left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs* **by** *fastforce*
moreover have *p* ∈ *poss* (*source A*)
using *assms rdp left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs* **by** *blast*
ultimately show *?thesis*
using Δ *left-lin-no-var-lhs.redex-patterns-single*[*OF ll-no-var-lhs*] **by** *blast*
qed

lemma *single-step-wf*:
assumes *A* ∈ *wf-pterm R* **and** Δ ∈ *set* (*single-steps A*)
shows Δ ∈ *wf-pterm R*

proof−
from *assms* **obtain** p α **where** p:$p \in$ *poss* (*source A*) Δ = *ll-single-redex* (*source A*) p α **and** *get-label* ((*labeled-source A*)|-p) = *Some* (α, *0*)
using *left-lin-no-var-lhs.redex-patterns-label left-lin-no-var-lhs.redex-patterns-subset-possL possL-subset-poss-source ll-no-var-lhs* **by** *fastforce*
then have *to-rule* $\alpha \in R$
using *assms*(*1*) *labeled-wf-pterm-rule-in-TRS* **by** *fastforce*
with p **show** *?thesis* **using** *single-redex-wf-pterm*
using *left-lin left-linear-trs-def* **by** *fastforce*
qed

lemma *source-single-step*:
assumes Δ:$\Delta \in$ *set* (*single-steps A*) **and** *wf*:$A \in$ *wf-pterm R*
shows *source* Δ = *source A*
proof−
let *?s*=*source A*
from Δ **obtain** p α **where** *pa*:Δ = *ll-single-redex ?s p* α (α, p) $\in$ *set* (*redex-patterns A*)
by *auto*
from *pa* **have** *lab-p*:*get-label* (*labeled-source A* |-p) = *Some* (α, *0*) **and** p:$p \in$ *poss ?s*
using *left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs wf* **by** *blast*+
from *lab-p p* **obtain** p' **where** p':$p' \in$*poss A* **and** *ctxt*:*ctxt-of-pos-term p* (*source A*) = *source-ctxt* (*ctxt-of-pos-term* p' *A*)
and *Ap'*:*A* |- p' = *Prule* α (*map* (λi. *A* |- (p' @ [*i*])) [*0*..<*length* (*var-rule* α)])
using *poss-labeled-source wf* **by** *force*
have *l*:*length* (*var-rule* α) = *length* (*var-poss-list* (*lhs* α))
using *wf* **by** (*metis Ap' Inl-inject Term.term.simps*(*4*) *is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *length-var-poss-list length-var-rule p' subt-at-is-wf-pterm term.inject*(*2*) *wf-pterm.simps*)
{fix *i* **assume** *i*:$i <$ *length* (*var-rule* α)
let *?pi*=*var-poss-list* (*lhs* α)!*i*
obtain x **where** x:*lhs* α |- *?pi* = *Var x var-rule* α!i = x
by (*metis comp-apply i l length-remdups-eq length-rev length-var-poss-list rev-rev-ident vars-term-list-var-poss-list*)
have *?s*|-p = *lhs* $\alpha \cdot \langle$*map source* (*map* (λi. *A* |- (p' @ [*i*])) [*0*..<*length* (*var-rule* α)])$\rangle_\alpha$
using *Ap' ctxt* **by** (*metis ctxt-of-pos-term-well ctxt-supt-id local.wf p p' replace-at-subt-at source.simps*(*3*) *source-ctxt-apply-term*)
moreover have *lhs* $\alpha \cdot \langle$*map source* (*map* (λi. *A* |- (p' @ [*i*])) [*0*..<*length* (*var-rule* α)])$\rangle_\alpha$ |- *?pi* = *map source* (*map* (λi. *A* |- (p' @ [*i*])) [*0*..<*length* (*var-rule* α)])!*i*
using x **by** (*smt* (*verit*, *ccfv-SIG*) *diff-zero eval-term.simps*(*1*) *i l length-upt lhs-subst-var-i map-eq-imp-length-eq map-nth nth-mem subt-at-subst var-poss-imp-poss var-poss-list-sound*)
ultimately have *?s*|-(*p*@*?pi*) = *source* (*A* |- (p' @ [*i*])) **using** *i p* **by** *auto*
then have *map source* (*map* (λi. *A* |- (p' @ [*i*])) [*0*..<*length* (*var-rule* α)]) !*i* = *map* (λpi. *source A* |- (*p* @ *pi*)) (*var-poss-list* (*lhs* α)) ! *i*
using *l i* **by** *auto*
}

with *l* **have** *map source* (*map* (λ*i*. *A* |- (*p'* @ [*i*])) [*0*..<*length* (*var-rule* α)]) = *map* (λ*pi*. *source A* |- (*p* @ *pi*)) (*var-poss-list* (*lhs* α))
by (*simp add*: *map-equality-iff*)
then have *source* (*A*|-*p'*) = *lhs* α · ⟨*map* (λ*pi*. *?s* |- (*p* @ *pi*)) (*var-poss-list* (*lhs* α))⟩α
unfolding *Ap' source.simps* **by** *simp*
with *ctxt* **show** *?thesis* **unfolding** *pa*(*1*) *source-single-redex*[*OF p*] **using** *p'*
by (*metis ctxt-of-pos-term-well ctxt-supt-id wf source-ctxt-apply-term*)
qed

lemma *single-redex-single-step*:
assumes Δ:Δ = *ll-single-redex s p* α
and *p*:*p* ∈ *poss s* **and** α:*to-rule* α ∈ *R*
and *src*:*source* Δ = *s*
shows *single-steps* Δ = [Δ]
using *src* **unfolding** Δ *left-lin-no-var-lhs.redex-patterns-single*[*OF ll-no-var-lhs p* α] **by** *simp*

lemma *single-step-label-imp-label*:
assumes Δ:Δ ∈ *set* (*single-steps A*) **and** *q*:*q* ∈ *poss* (*labeled-source* Δ) **and** *wf*:*A* ∈ *wf-pterm R*
and *lab*:*get-label* (*labeled-source* Δ|-*q*) = *Some l*
shows *get-label* (*labeled-source A* |-*q*) = *Some l*
proof−
let *?s*=*source A*
from Δ **obtain** *p* α **where** *pa*:Δ = *ll-single-redex ?s p* α (α, *p*) ∈ *set* (*redex-patterns A*)
by *auto*
from *pa* **have** *lab-p*:*get-label* (*labeled-source A* |-*p*) = *Some* (α, *0*) **and** *p*:*p* ∈ *poss* (*source A*)
using *left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs wf* **by** *blast*+
from *pa lab* **obtain** *q'* **where** *l*:*l* = (α, *size q'*) **and** *q'*:*q* = *p*@*q' q'* ∈ *fun-poss* (*lhs* α)
using *single-redex-label*[*OF pa*(*1*) *p*] *q pa*(*2*) *wf*
by (*metis labeled-source-to-term labeled-wf-pterm-rule-in-TRS left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs poss-term-lab-to-term prod.collapse*)
from *lab-p p* **obtain** *p'* **where** *p'*∈*poss A* **and** *ctxt-of-pos-term p* (*source A*) = *source-ctxt* (*ctxt-of-pos-term p' A*) **and** *A* |- *p'* = *Prule* α (*map* (λ*i*. *A* |- (*p'* @ [*i*])) [*0*..<*length* (*var-rule* α)])
using *poss-labeled-source wf* **by** *force*
then have *labeled-source A* = (*ctxt-of-pos-term p* (*labeled-source A*))⟨*labeled-source* (*Prule* α (*map* (λ*i*. *A* |- (*p'* @ [*i*])) [*0*..<*length* (*var-rule* α)]))⟩
using *label-source-ctxt p wf* **by** (*metis ctxt-supt-id*)
then have *labeled-source A*|-*q* = *labeled-lhs* α · ⟨*map labeled-source* (*map* (λ*i*. *A* |- (*p'* @ [*i*])) [*0*..<*length* (*var-rule* α)])⟩α |-*q'*
unfolding *q' labeled-source.simps* **by** (*metis labeled-source.simps*(*3*) *labeled-source-to-term p poss-term-lab-to-term subt-at-append subt-at-ctxt-of-pos-term*)
then have *get-label* (*labeled-source A*|-*q*) = *Some* (α, *size q'*)
using *q'*(*2*) **by** (*simp add*: *label-term-increase*)

with *l* **show** *?thesis* **by** *simp*
qed

lemma *single-steps-measure*:
assumes $\Delta 1$:$\Delta 1 \in$ *set* (*single-steps A*) **and** $\Delta 2$:$\Delta 2 \in$ *set* (*single-steps A*)
and *wf*:*A* ∈ *wf-pterm R* **and** *neq*:$\Delta 1 \neq \Delta 2$
shows *measure-ov* $\Delta 1$ $\Delta 2$ = *0*
proof−
let *?s*=*source A*
from $\Delta 1$ **obtain** *p* α **where** *pa*:$\Delta 1$ = *ll-single-redex ?s p* α (α, *p*) ∈ *set* (*redex-patterns A*)
by *auto*
from $\Delta 2$ **obtain** *q* β **where** *qb*:$\Delta 2$ = *ll-single-redex ?s q* β (β, *q*) ∈ *set* (*redex-patterns A*)
by *auto*
from *neq* **have** *pq*:*p* ≠ *q* ∨ $\alpha \neq \beta$
using *pa*(*1*) *qb*(*1*) **by** *force*
{assume *measure-ov* $\Delta 1$ $\Delta 2 \neq$ *0*
then obtain *r* **where** *r1*:*r* ∈ *possL* $\Delta 1$ **and** *r2*:*r* ∈ *possL* $\Delta 2$
by (*metis card.empty disjoint-iff*)
from *r1* **obtain** *p*′ **where** *p*′:*r* = *p*@*p*′ **and** *l1*:*get-label* (*labeled-source* $\Delta 1$ |-*r*) = *Some* (α, *size p*′)
using *single-redex-label*[*OF pa*(*1*)] *wf*
by (*smt* (*verit*, *ccfv-SIG*) *labeled-source-to-term labeled-wf-pterm-rule-in-TRS left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs pa*(*2*) *possL-obtain-label possL-subset-poss-source poss-term-lab-to-term subsetD*)
from *r2* **obtain** *q*′ **where** *q*′:*r* = *q*@*q*′ **and** *l2*:*get-label* (*labeled-source* $\Delta 2$ |-*r*) = *Some* (β, *size q*′)
using *single-redex-label*[*OF qb*(*1*)] *wf*
by (*smt* (*verit*, *ccfv-SIG*) *labeled-source-to-term labeled-wf-pterm-rule-in-TRS left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs qb*(*2*) *possL-obtain-label possL-subset-poss-source poss-term-lab-to-term subsetD*)
from *l1* **have** *get-label* (*labeled-source A* |-*r*) = *Some* (α, *size p*′)
using $\Delta 1$ *labelposs-subs-poss wf r1 single-step-label-imp-label* **by** *blast*
moreover from *l2* **have** *get-label* (*labeled-source A* |-*r*) = *Some* (β, *size q*′)
using $\Delta 2$ *labelposs-subs-poss wf r2 single-step-label-imp-label* **by** *blast*
moreover from *pq* **have** *p*′ ≠ *q*′ ∨ $\alpha \neq \beta$
using *p*′ *q*′ **by** *blast*
ultimately have *False* **using** *p*′ *q*′ **by** *auto*
}
then show *?thesis* **by** *auto*
qed

lemma *single-steps-orth*:
assumes $\Delta 1$:$\Delta 1 \in$ *set* (*single-steps A*) **and** $\Delta 2$:$\Delta 2 \in$ *set* (*single-steps A*) **and** *wf*:*A* ∈ *wf-pterm R*
shows $\Delta 1$ $\perp_p$ $\Delta 2$
using *single-steps-measure*[*OF* $\Delta 1$ $\Delta 2$ *wf*] *equal-imp-orthogonal*
by (*metis* $\Delta 1$ $\Delta 2$ *ll-no-var-lhs local.wf measure-zero-imp-orthogonal single-step-wf*

source-single-step)

lemma *redex-patterns-below*:
assumes *wf*:$A \in$ *wf-pterm* R
and (α, p) $\in$ *set* (*redex-patterns* A)
and (β, p@q) $\in$ *set* (*redex-patterns* A) **and** $q \neq []$
shows $q \notin$ *fun-poss* (*lhs* α)
proof−
let *?Δ1*=*ll-single-redex* (*source* A) p α
let *?Δ2*=*ll-single-redex* (*source* A) (p@q) β
from *assms* **have** *Δ1*:*?Δ1* $\in$ *set* (*single-steps* A)
by *force*
from *assms* **have** *Δ2*:*?Δ2* $\in$ *set* (*single-steps* A)
by *force*
from *assms*(*1*,*2*) **have** *possL1*:*possL ?Δ1* = $\{p@p' \mid p'.\ p' \in$ *fun-poss* (*lhs* α)$\}$
by (*metis* (*no-types*, *lifting*) *left-lin-no-var-lhs.redex-pattern-rule-symbol left-lin-no-var-lhs.redex-patterns-lab*
ll-no-var-lhs single-redex-possL)
from *assms*(*1*,*3*) **have** *possL2*:*possL ?Δ2* = $\{(p@q)@p' \mid p'.\ p' \in$ *fun-poss* (*lhs*
β)$\}$
using *left-lin.single-redex-possL left-lin-axioms left-lin-no-var-lhs.redex-pattern-rule-symbol*
left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs **by** *blast*
from *assms* **have** *neq*:*?Δ1* $\neq$ *?Δ2*
by (*metis Pair-inject left-lin-no-var-lhs.redex-patterns-label ll-no-var-lhs self-append-conv*
single-redex-neq)
from *single-steps-measure*[*OF Δ1 Δ2 wf neq*] **have** *possL ?Δ1* $\cap$ *possL ?Δ2* =
$\{\}$
by (*simp add*: *finite-possL*)
moreover have $[] \in$ *fun-poss* (*lhs* β) **proof**−
have *to-rule* $\beta \in R$
using *assms*(*1*) *assms*(*3*) *left-lin-no-var-lhs.redex-pattern-rule-symbol ll-no-var-lhs*
by *blast*
then show *?thesis*
using *wf-trs-alt wf-trs-imp-lhs-Fun* **by** *fastforce*
qed
ultimately show *?thesis*
unfolding *possL1 possL2* **by** *auto*
qed

lemma *single-steps-singleton*:
assumes *A-wf*:$A \in$ *wf-pterm* R **and** Δ:*single-steps* $A = [\Delta]$
shows $A = \Delta$
proof−
obtain p α **where** *rdp-Δ*:Δ = *ll-single-redex* (*source* A) p α (α, p) $\in$ *set*
(*redex-patterns* A) *redex-patterns* $\Delta = [(\alpha, p)]$
using *single-step-redex-patterns*[*OF A-wf*] Δ **by** *auto*
then have *rdp-A*:*redex-patterns* $A = [(\alpha, p)]$
by (*smt* (*verit*) Δ *in-set-simps*(*2*) *list.map-disc-iff map-eq-Cons-D*)
then show *?thesis*
using *left-lin-no-var-lhs.redex-pattern-proof-term-equality*[*OF ll-no-var-lhs A-wf*]

by (*metis A-wf* Δ *list.set-intros*(*1*) *rdp-*Δ(*3*) *single-step-wf source-single-step*)
qed
end

context *left-lin-no-var-lhs*
begin
lemma *measure-ov-imp-single-step-ov*:
assumes *measure-ov A B* ≠ *0* **and** *wf*:*A* ∈ *wf-pterm R*
shows ∃Δ ∈ *set* (*single-steps A*). *measure-ov* Δ *B* ≠ *0*
proof−
from *assms* **obtain** *r* **where** *r1*:*r* ∈ *possL A* **and** *r2*:*r* ∈ *possL B*
by (*metis card.empty disjoint-iff*)
then obtain α *n* **where** *lab*:*get-label* (*labeled-source A* |- *r*) = *Some* (α, *n*)
using *possL-obtain-label* **by** *blast*
with *wf r1* **obtain** *r1 r2* **where** *r*:*r* = *r1*@*r2* **and** *lab-r1*:*get-label* (*labeled-source A* |-*r1*) = *Some* (α, *0*) **and** *n*:*length r2* = *n*
by (*metis* (*no-types*, *lifting*) *append-take-drop-id diff-diff-cancel label-term-max-value labelposs-subs-poss length-drop obtain-label-root subsetD*)
from *r r1* **have** *r1-pos*:*r1* ∈ *poss* (*labeled-source A*)
using *labelposs-subs-poss poss-append-poss* **by** *blast*
then obtain *q* **where** *q*:*q*∈*poss A* **and** *ctxt*:*ctxt-of-pos-term r1* (*source A*) = *source-ctxt* (*ctxt-of-pos-term q A*)
and *Aq*:*A* |- *q* = *Prule* α (*map* (λ*i*. *A* |- (*q* @ [*i*])) [*0*..<*length* (*var-rule* α)])
using *poss-labeled-source wf lab-r1* **by** *blast*
with *r r1* **have** *r2-pos*:*r2* ∈ *poss* (*source* (*Prule* α (*map* (λ*i*. *A* |- (*q* @ [*i*])) [*0*..<*length* (*var-rule* α)])))
by (*metis* (*no-types*, *lifting*) *ctxt-supt-id fun-poss-imp-poss label-source-ctxt labeled-source-to-term labelposs-subs-fun-poss-source local.wf poss-term-lab-to-term r1-pos replace-at-subt-at subterm-poss-conv*)
from *Aq q wf* **have** *Prule* α (*map* (λ*i*. *A* |- (*q* @ [*i*])) [*0*..<*length* (*var-rule* α)]) ∈ *wf-pterm R*
using *subt-at-is-wf-pterm* **by** *auto*
moreover then have *is-Fun* (*lhs* α) **using** *no-var-lhs*
using *wf-pterm.cases* **by** *fastforce*
moreover from *lab ctxt* **have** *get-label* (*labeled-source* (*Prule* α (*map* (λ*i*. *A* |- (*q* @ [*i*])) [*0*..<*length* (*var-rule* α)])) |-*r2*) = *Some* (α, *n*)
by (*metis* (*no-types*, *lifting*) *Aq ctxt-supt-id label-source-ctxt labeled-source-to-term local.wf poss-term-lab-to-term q r r1-pos replace-at-subt-at subt-at-append*)
ultimately have *r2-funposs*:*r2* ∈ *fun-poss* (*lhs* α)
using *labeled-poss-in-lhs*[*OF r2-pos*] *n* **by** *blast*
let *?*Δ=*ll-single-redex* (*source A*) *r1* α
from *lab-r1 r1-pos* **have** *rdp*:(α, *r1*) ∈ *set* (*redex-patterns A*)
using *redex-patterns-label wf* **by** *auto*
then have Δ:*?*Δ ∈ *set* (*single-steps A*) **by** *force*
from *r2* **have** *measure-ov ?*Δ *B* ≠ *0*
by (*smt* (*verit*, *ccfv-threshold*) *rdp labeled-sources-imp-measure-not-zero labeled-wf-pterm-rule-in-TRS labelposs-subs-poss wf mem-Collect-eq option.simps*(*3*) *possL-obtain-label r r1-pos r2-funposs redex-patterns-label rel-simps*(*70*) *single-redex-possL subsetD*)

with Δ **show** *?thesis* **by** *blast*
qed
end

context *left-lin-no-var-lhs*
begin
lemma *label-single-step*:
assumes $p \in$ *poss* (*labeled-source A*) $A \in$ *wf-pterm R*
and *get-label* (*labeled-source A* |- *p*) = *Some* (α, *n*)
shows $\exists$ *Ai*. *Ai* $\in$ *set* (*single-steps A*) $\wedge$ *get-label* (*labeled-source Ai* |- *p*) = *Some* (α, *n*)
proof−
let *?p1*=*take* (*length p* − *n*) *p*
let *?p2*=*drop* (*length p* − *n*) *p*
let *?xs*=*map* (*to-pterm* ∘ (λ*pi*. (*source A*)|-(*p*@*pi*))) (*var-poss-list* (*lhs* α))
from *assms*(*1*) **have** *p1-pos*:*?p1* $\in$ *poss* (*labeled-source A*)
by (*metis append-take-drop-id poss-append-poss*)
have *lab*:*get-label* (*labeled-source A* |- *?p1*) = *Some* (α, *0*)
using *obtain-label-root*[*OF assms*(*1*) *assms*(*3*) *assms*(*2*)] **by** *simp*
with *assms* **have** *rdp*:(α, *?p1*) $\in$ *set* (*redex-patterns A*)
using *redex-patterns-label*[*OF assms*(*2*)] **by** (*metis labeled-source-to-term obtain-label-root poss-term-lab-to-term*)
then have *ll-single-redex* (*source A*) *?p1* α $\in$ *set* (*single-steps A*) **by** *force*
then obtain *Ai* **where** *Ai*:*Ai* $\in$ *set* (*single-steps A*) *Ai* = *ll-single-redex* (*source A*) *?p1* α
by *presburger*
from *rdp* **obtain** p' *As* **where** p':*A*|-p' = *Prule* α *As* $p' \in$ *poss A ctxt-of-pos-term ?p1* (*source A*) = *source-ctxt* (*ctxt-of-pos-term* p' *A*)
using *poss-labeled-source*[*OF p1-pos*] *lab assms*(*2*) **by** *blast*
from p' *assms*(*2*) **have** *A*|-p' $\in$ *wf-pterm R*
using *subt-at-is-wf-pterm* **by** *blast*
moreover from p' *assms* **have** *get-label* (*labeled-source* (*A*|-p') |- *?p2*) = *Some* (α, *n*)
by (*smt* (*verit*, *ccfv-SIG*) *append-take-drop-id ctxt-supt-id label-source-ctxt p1-pos rdp redex-patterns-label replace-at-subt-at subterm-poss-conv*)
ultimately have *p2-pos*:*?p2* $\in$ *fun-poss* (*lhs* α)
using *labeled-poss-in-lhs no-var-lhs assms* p'
by (*smt* (*verit*, *ccfv-threshold*) *append-take-drop-id case-prod-conv ctxt-of-pos-term-well ctxt-supt-id diff-diff-cancel label-term-max-value labeled-source-to-term labeled-wf-pterm-rule-in-TRS length-drop poss-append-poss poss-term-lab-to-term replace-at-subt-at source-ctxt-apply-term*)
then have *l*:*get-label* (*labeled-source* (*Prule* α *?xs*) |- *?p2*) = *Some* (α, *n*)
using *label-term-increase assms* **by** (*metis* (*no-types*, *lifting*) *add-0 diff-diff-cancel label-term-max-value labeled-source.simps*(*3*) *length-drop*)
from *p1-pos* **have** *?p1* $\in$ *poss* (*source A*) **by** *simp*
then have *get-label* (*labeled-source Ai* |- *p*) = *Some* (α, *n*)
unfolding *Ai*(*2*) **by** (*metis p2-pos append-take-drop-id l label-ctxt-apply-term label-term-increase labeled-source.simps*(*3*) *ll-single-redex-def*)
with *Ai* **show** *?thesis*

by *blast*
qed

lemma *proof-term-matches*:
assumes $A \in$ *wf-pterm R B* $\in$ *wf-pterm R linear-term A*
and $\bigwedge\alpha$ *r*. $(\alpha, r) \in$ *set* (*redex-patterns A*) = ($(\alpha, r) \in$ *set* (*redex-patterns B*) $\wedge$ *r* $\in$ *fun-poss* (*source A*))
and *source A* $\cdot$ σ = *source B*
shows *A* $\cdot$ (*mk-subst Var* (*match-substs A B*)) = *B*
proof−
{fix *p g ts* **assume** *p* $\in$ *poss A A*|-*p* = *Fun g ts*
with *assms* **have** $\exists$ *Bs*. *length ts* = *length Bs* $\wedge$ *B*|-*p* = *Fun g Bs*
using *assms* **proof**(*induct A arbitrary*: *B p rule*:*pterm-induct*)
case (*Pfun f As*)
then have $\bigwedge\alpha$. $(\alpha, [])$ $\notin$ *set* (*redex-patterns* (*Pfun f As*))
by (*meson list.distinct*(*1*) *redex-patterns-elem-fun*)
with *Pfun*(*5*) **have** ¬ (*is-Prule B*)
by (*metis empty-pos-in-poss is-FunI is-Prule.elims*(*2*) *list.set-intros*(*1*) *poss-is-Fun-fun-poss redex-patterns.simps*(*3*) *source.simps*(*2*) *subt-at.simps*(*1*))
with *Pfun*(*6*) **obtain** *Bs* **where** *B*:*B* = *Pfun f Bs* **and** *l*:*length Bs* = *length As*
by (*smt* (*verit*, *del-insts*) *eval-term.simps*(*2*) *is-Prule.elims*(*3*) *length-map source.simps*(*1*) *source.simps*(*2*) *term.distinct*(*1*) *term.inject*(*2*))
then show *?case* **proof**(*cases p*)
case *Nil*
from *Pfun*(*8*) **show** *?thesis* **unfolding** *Nil B* **using** *l* **by** *simp*
next
case (*Cons i p*′)
from *Pfun*(*7*) **have** *i*:*i* < *length As* **and** *p*′:*p*′ $\in$ *poss* (*As*!*i*) **and** *a*:*As*!*i* $\in$ *set As*
unfolding *Cons* **by** *simp-all*
from *Pfun*(*8*) **have** *at-p*′:(*As*!*i*)|-*p*′ = *Fun g ts*
unfolding *Cons* **by** *simp*
from *Pfun*(*2*) **have** *a-wf*:*As*!*i* $\in$ *wf-pterm R*
using *i nth-mem* **by** *blast*
from *Pfun*(*3*) **have** *b-wf*:*Bs*!*i* $\in$ *wf-pterm R*
unfolding *B* **using** *i l* **by** *auto*
from *Pfun*(*4*) **have** *a-lin*:*linear-term* (*As*!*i*)
using *i* **by** *simp*
{fix α *r* **assume** $(\alpha, r) \in$ *set* (*redex-patterns* (*As*!*i*))
then have $(\alpha, i\#r) \in$ *set* (*redex-patterns* (*Pfun f As*))
by (*meson i redex-patterns-elem-fun*′)
with *Pfun*(*5*) **have** $(\alpha, r) \in$ *set* (*redex-patterns* (*Bs*!*i*)) $\wedge$ $i\#r \in$ *fun-poss* (*source* (*Pfun f As*))
unfolding *B* **by** (*metis list.inject redex-patterns-elem-fun*)
then have $(\alpha, r) \in$ *set* (*redex-patterns* (*Bs*!*i*)) $\wedge$ *r* $\in$ *fun-poss* (*source* (*As*!*i*))
using *i* **by** *simp*
} moreover {fix α *r* **assume** $(\alpha, r) \in$ *set* (*redex-patterns* (*Bs*!*i*)) **and** *r*:*r*

```
∈ fun-poss (source (As!i))
        then have (α, i#r) ∈ set (redex-patterns B)
          unfolding B using i l by (metis redex-patterns-elem-fun')
        moreover from r have i#r ∈ fun-poss (source (Pfun f As))
          using i unfolding source.simps fun-poss.simps length-map by simp
        ultimately have (α, r) ∈ set (redex-patterns (As!i))
          using Pfun(5) i by (metis list.inject redex-patterns-elem-fun)
      }
      ultimately have rdp:⋀α r. ((α, r) ∈ set (redex-patterns (As ! i))) = ((α,
r) ∈ set (redex-patterns (Bs ! i)) ∧ r ∈ fun-poss (source (As ! i)))
        by blast
      from Pfun(6) have sigma:source (As!i) · σ = source (Bs!i)
        unfolding B source.simps eval-term.simps using i l using map-nth-conv
by fastforce
       from Pfun(1)[OF a a-wf b-wf a-lin rdp sigma p' at-p' a-wf b-wf a-lin rdp
sigma]
        obtain ss where length ts = length ss and Bs!i |- p' = Fun g ss by blast
      then show ?thesis unfolding B Cons using i l by simp
    qed
  next
    case (Prule α As)
    from Prule(5) have (α, []) ∈ set (redex-patterns B)
      by auto
    then obtain Bs where B:B = Prule α Bs
      using [[metis-instantiate]]
    by (metis Prule.prems(2) in-set-idx in-set-simps(3) redex-patterns-elem-fun[of
α []]
          less-nat-zero-code list.distinct(1) nat-neq-iff nth-Cons-0
          prefix-order.dual-order.refl[of []] prod.inject redex-patterns.simps(1)
       redex-patterns.simps(3) redex-patterns-order[of B 0 - - [] α []] wf-pterm.simps)
    with Prule(2,3) have l:length As = length Bs
      using length-args-well-Prule by blast
    show ?case proof(cases p)
      case Nil
      from Prule(8) show ?thesis unfolding Nil B using l by simp
    next
      case (Cons i p')
      from Prule(7) have i:i < length As and p':p' ∈ poss (As!i) and a:As!i ∈
set As
        unfolding Cons by simp-all
      from Prule(8) have at-p':(As!i)|-p' = Fun g ts
        unfolding Cons by simp
      from Prule(2) have a-wf:As!i ∈ wf-pterm R
        using i nth-mem by blast
      from Prule(3) have b-wf:Bs!i ∈ wf-pterm R
        unfolding B using i l by auto
      from Prule(4) have a-lin:linear-term (As!i)
        using i by simp
      let ?pi=var-poss-list (lhs α) ! i
```

let *?xi=vars-term-list* (*lhs* α) ! *i*
have *i'*:*i* < *length* (*var-poss-list* (*lhs* α))
using *i Prule*(*2*) **by** (*metis Inl-inject is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *length-var-poss-list length-var-rule term.distinct*(*1*) *term.inject*(*2*) *wf-pterm.simps*)

have *eval-lhs'*:$\bigwedge\sigma$. *lhs* α · σ |- *?pi* = σ *?xi*
by (*metis eval-term.simps*(*1*) *i'* *length-var-poss-list nth-mem subt-at-subst var-poss-imp-poss var-poss-list-sound vars-term-list-var-poss-list*)
then have *eval-lhs*:$\bigwedge\sigma$ *q*. *lhs* α · σ |- (*?pi*@*q*) = σ *?xi* |- *q*
by (*smt* (*verit*) *i'* *nth-mem poss-imp-subst-poss subt-at-append var-poss-imp-poss var-poss-list-sound*)
have *i* < *length* (*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map redex-patterns Bs*))
unfolding *length-map length-zip* **using** *i l i'* **by** *simp*
moreover have *zip* (*var-poss-list* (*lhs* α)) (*map redex-patterns Bs*) ! *i* = (*?pi*, *redex-patterns* (*Bs*!*i*))
using *i i' l* **by** *force*
ultimately have *map-rdp*:(*map2* (λ*p1*. *map* (λ(α, *p2*). (α, *p1* @ *p2*))) (*var-poss-list* (*lhs* α)) (*map redex-patterns Bs*))!*i* = *map* (λ(α, *p2*). (α, *?pi* @ *p2*)) (*redex-patterns* (*Bs*!*i*))
by *simp*
have *l'*:*length* (*var-rule* α) = *length* (*vars-term-list* (*lhs* α))
using *B Prule.prems*(*2*) *left-lin.length-var-rule left-lin-axioms wf-pterm.simps* **by** *fastforce*
{fix β *r* **assume** β*r*:(β, *r*) ∈ *set* (*redex-patterns* (*As*!*i*))
from *i'* **have** (β, *?pi*@*r*) ∈ *set* (*redex-patterns* (*Prule* α *As*))
using *redex-patterns-elem-rule'*[*OF* β*r i*] **by** *simp*
with *Prule*(*5*) **have** *1*:(β, *?pi*@*r*) ∈ *set* (*redex-patterns B*) **and** *2*:*?pi*@*r* ∈ *fun-poss* (*source* (*Prule* α *As*))
by *presburger+*
from *1* **have** (β, *r*) ∈ *set* (*redex-patterns* (*Bs*!*i*))
using *redex-patterns-rule''* **by** (*metis B Prule.prems*(*2*) *i l*)
moreover have *r* ∈ *fun-poss* (*source* (*As*!*i*))
by (*metis* β*r a-wf get-label-imp-labelposs labeled-source-to-term label-poss-subs-fun-poss-source left-lin-no-var-lhs.redex-patterns-label left-lin-no-var-lhs-axioms option.distinct*(*1*) *poss-term-lab-to-term*)
ultimately have (β, *r*) ∈ *set* (*redex-patterns* (*Bs*!*i*)) ∧ *r* ∈ *fun-poss* (*source* (*As*!*i*))
by *simp*
} moreover {fix β *r* **assume** β*r*:(β, *r*) ∈ *set* (*redex-patterns* (*Bs*!*i*)) **and** *r*:*r* ∈ *fun-poss* (*source* (*As*!*i*))
let *?x=var-rule* α ! *i*
from *l'* **have** *x*:*lhs* α |- *?pi* = *Var ?x*
using *i* **by** (*metis comp-apply eval-lhs' length-remdups-eq length-rev rev-rev-ident subst-apply-term-empty*)
with *r* **have** *r*:*r* ∈ *fun-poss* (*lhs* α |- *?pi* · ⟨*map source As*⟩$_\alpha$)
using *lhs-subst-var-i l' i i'* **by** (*metis* (*mono-tags*, *lifting*) *eval-term.simps*(*1*) *length-map length-var-poss-list nth-map*)
from β*r* **have** (β, *?pi*@*r*) ∈ *set* (*redex-patterns B*)

unfolding B **using** i l **using** *redex-patterns-elem-rule'*[*OF* βr i[*unfolded* l] i'] **by** *simp*
moreover from r x **have** *?pi@r* ∈ *fun-poss* (*source* (*Prule* α *As*))
using i **unfolding** *source.simps fun-poss.simps*
by (*metis* (*no-types*, *lifting*) *eval-lhs eval-lhs' fun-poss-fun-conv fun-poss-imp-poss* i' *is-FunI nth-mem*
pos-append-poss poss-imp-subst-poss poss-is-Fun-fun-poss subt-at-subst var-poss-imp-poss var-poss-list-sound)
ultimately have (β, *?pi* @ r) ∈ *set* (*redex-patterns* (*Prule* α *As*))
using *Prule*(*5*) **by** *presburger*
then have (β, r) ∈ *set* (*redex-patterns* (*As*!i))
using i *redex-patterns-rule''* *Prule.prems*(*1*) **by** *blast*
}
ultimately have *rdp*:⋀α r. ((α, r) ∈ *set* (*redex-patterns* (*As* ! i))) = ((α, r) ∈ *set* (*redex-patterns* (*Bs* ! i)) ∧ r ∈ *fun-poss* (*source* (*As* ! i)))
by *blast*
from *Prule*(*6*) **have** *sigma*:*source* (*As*!i) ⋅ σ = *source* (*Bs*!i)
unfolding B *source.simps eval-term.simps* **using** i l *map-nth-conv*
by (*smt* (*verit*, *best*) B *Inl-inject Prule.prems*(*2*) *apply-lhs-subst-var-rule comp-apply eval-lhs' i' is-Prule.simps*(*1*) *is-Prule.simps*(*3*) *length-map length-remdups-eq length-rev length-var-rule nth-mem poss-imp-subst-poss rev-swap subt-at-subst term.distinct*(*1*) *term.inject*(*2*) *var-poss-imp-poss var-poss-list-sound wf-pterm.simps*)
from *Prule*(*1*)[*OF a a-wf b-wf a-lin rdp sigma p' at-p' a-wf b-wf a-lin rdp sigma*]
obtain *ss* **where** *length ts* = *length ss* **and** *Bs*!i |- p' = *Fun g ss* **by** *blast*
then show *?thesis* **unfolding** B *Cons* **using** i l **by** *simp*
qed
qed *simp*
}
then show *?thesis* **using** *fun-poss-eq-imp-matches*[*OF assms*(*3*)] **by** *simp*
qed
end

context *left-lin-wf-trs*
begin
lemma *join-single-steps-wf*:
assumes A ∈ *wf-pterm R*
and *As* = *filter f* (*single-steps A*) **and** *As* ≠ []
shows ∃ D. *join-list As* = *Some D* ∧ D ∈ *wf-pterm R*
proof−
{**fix** *a1 a2* **assume** *a1*:*a1* ∈ *set* (*single-steps A*) **and** *a2*:*a2* ∈ *set* (*single-steps A*)
with *assms*(*1*,*2*) **have** *a1* ⊥$_p$ *a2* ∨ *a1* = *a2*
using *single-steps-orth* **by** *presburger*
moreover from *a1* **have** *a1* ∈ *wf-pterm R*
using *single-step-wf*[*OF assms*(*1*)] *assms*(*2*) **by** *presburger*
moreover from *a2* **have** *a2* ∈ *wf-pterm R*
using *single-step-wf*[*OF assms*(*1*)] *assms*(*2*) **by** *presburger*
ultimately have *a1* ⊔ *a2* ≠ *None*

```
    using join-same orth-imp-join-defined no-var-lhs by fastforce
  }
  then show ?thesis using left-lin-no-var-lhs.join-list-defined[OF ll-no-var-lhs]
assms(2,3) single-step-wf[OF assms(1)] by simp
qed

lemma single-steps-join-list:
  assumes join-list As = Some A and ∀ a ∈ set As. a ∈ wf-pterm R
  shows set (single-steps A) = ⋃ (set (map (set ∘ single-steps) As))
proof−
  have rdp:set (redex-patterns A) = ⋃ (set (map (set ∘ redex-patterns) As))
    using left-lin-no-var-lhs.redex-patterns-join-list assms ll-no-var-lhs by blast
  {fix a assume a ∈ set (single-steps A)
   then obtain α p where a:a = ll-single-redex (source A) p α and (α, p) ∈ set
(redex-patterns A) by auto
    with rdp obtain Ai where Ai:Ai ∈ set As and (α, p) ∈ set (redex-patterns
Ai) by auto
    then have a ∈ set (single-steps Ai)
     unfolding a using left-lin-no-var-lhs.source-join-list[OF ll-no-var-lhs assms]
by force
    with Ai have a ∈ ⋃ (set (map (set ∘ single-steps) As)) by auto
  } moreover
  {fix a assume a ∈ ⋃ (set (map (set ∘ single-steps) As))
   then obtain Ai where Ai:Ai ∈ set As a ∈ set (single-steps Ai)
     by (smt (verit, best) UnionE comp-def in-set-idx length-map map-nth-eq-conv
nth-mem)
   then obtain α p where a:a = ll-single-redex (source Ai) p α and (α, p) ∈ set
(redex-patterns Ai) by auto
    with rdp Ai have (α, p) ∈ set (redex-patterns A) by auto
    then have a ∈ set (single-steps A)
     unfolding a using left-lin-no-var-lhs.source-join-list[OF ll-no-var-lhs assms]
Ai by force
  }
  ultimately show ?thesis by fastforce
qed
end

end
```

References

[1] C. Kirk and A. Middeldorp. Formalizing simultaneous critical pairs for confluence of left-linear rewrite systems. In *Proc. 14th International Conference on Certified Programs and Proofs*, pages 156–170, 2025.

[2] C. Kohl and A. Middeldorp. A formalization of the development closedness criterion for left-linear term rewrite systems. In *Proc. 12th Inter-*

national Conference on Certified Programs and Proofs, pages 197–210, 2023.

[3] C. Kohl and A. Middeldorp. Formalizing almost development closed critical pairs (short paper). In *Proc. 14th International Joint Conference on Automated Reasoning*, volume 268 of *LIPIcs*, pages 38:1–38:8, 2023.

[4] C. Kohl and A. Middeldorp. Formalizing confluence and commutation criteria using proof terms. In *Proc. 12th International Workshop on Confluence*, pages 49–54, 2023. Available from http://cl-informatik.uibk.ac.at/iwc/2023/proceedings.pdf.

[5] TeReSe, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[6] V. van Oostrom and R. de Vrijer. Four equivalent equivalences of reductions. In *Proc. 2nd International Workshop on Reduction Strategies in Rewriting and Programming*, volume 70(6) of *Electronic Notes in Theoretical Computer Science*, pages 21–61, 2002.