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* . . . 4
- 1.2 Results About Linear Terms . . . 4
- 1.3 Results About Substitutions and Contexts . . . 5
  - 1.3.1 Utilities for *mk-subst* . . . 7
- 1.4 Matching Terms . . . 7
  - 1.4.1 Matching of Linear Terms . . . 8

2 Proof Terms **9**
- 2.1 Definitions . . . 9
- 2.2 Frequently Used Locales/Contexts . . . 11
- 2.3 Proof Term Predicates . . . 12
- 2.4 'Normal' Terms vs. Proof Terms . . . 14

2.5 Substitutions . . . 15
2.6 Contexts . . . 16
2.7 Source and Target . . . 17
2.8 Additional Results . . . 19
2.9 Proof Terms Represent Multi-Steps . . . 20

3 Operations on Proof Terms **20**
3.1 Residuals . . . 21
3.2 Join . . . 25
3.2.1 N-Fold Join . . . 28
3.3 Deletion . . . 30
3.4 Computations With Single Redexes . . . 30

4 Orthogonal Proof Terms **34**

5 Labels and Overlaps **35**
5.1 Labeled Proof Terms . . . 36
5.2 Measuring Overlap . . . 44
5.3 Collecting Overlapping Positions . . . 45

6 Redex Patterns **47**

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 $\forall\, x \in set\ xs.\ \exists\, a.\ P\ a\ x$
shows $\exists\, as.\ length\ as = length\ xs \wedge (\forall\, i < length\ xs.\ P\ (as!i)\ (xs!i))$
⟨*proof*⟩

lemma *card-Union-Sum*:
assumes *is-partition* (*map f* [*0*..<*length xs*])
and $\forall\, i < length\ xs.\ finite\ (f\ i)$
shows *card* $(\bigcup i{<}length\ xs.\ f\ i) = (\sum i{<}length\ xs.\ card\ (f\ i))$
⟨*proof*⟩

lemma *sum-sum-concat*: $(\sum i{<}length\ xs.\ \sum x{\leftarrow}f\ (xs!i).\ g\ x) = (\sum x{\leftarrow}concat\ (map\ f\ xs).\ g\ x)$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *zip-symm*: (*x*, *y*) ∈ *set* (*zip xs ys*) ⟹ (*y*, *x*) ∈ *set* (*zip ys xs*)
⟨*proof*⟩

lemma *sum-list-elem*:
(∑ *x*←[*y*]. *f x*) = *f y*
⟨*proof*⟩

lemma *sum-list-zero*:
assumes ∀ *i* < *length xs*. *f* (*xs*!*i*) = *0*
shows (∑ *x*←*xs*. *f x*) = *0*
⟨*proof*⟩

lemma *distinct-is-partition*:
assumes *distinct* (*concat ts*)
shows *is-partition* (*map set ts*)
⟨*proof*⟩

lemma *filter-ex-index*:
assumes *x* = *filter f xs* ! *i i* < *length* (*filter f xs*)
shows ∃ *j*. *j* < *length xs* ∧ *x* = *xs*!*j*
⟨*proof*⟩

lemma *filter-index-neq'*:
assumes *i* < *j 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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

1.1.1 Lists of *option*

lemma *length-those*:
assumes *those xs* = *Some ys*
shows *length xs* = *length ys*
⟨*proof*⟩

lemma *those-not-none-x*: *those xs* = *Some ys* ⟹ *x* ∈ *set xs* ⟹ *x* ≠ *None*
⟨*proof*⟩

lemma *those-not-none-xs*: *list-all* (λ*x*. *x* ≠ *None*) *xs* ⟹ *those xs* ≠ *None*
⟨*proof*⟩

lemma *those-some*:
assumes *length xs* = *length ys* ∀ *i* < *length xs*. *xs*!*i* = *Some* (*ys*!*i*)
shows *those xs* = *Some ys*
⟨*proof*⟩

lemma *those-some2*:
assumes *those xs* = *Some ys*
shows ∀ *i* < *length xs*. *xs*!*i* = *Some* (*ys*!*i*)
⟨*proof*⟩

lemma *exists-some-list*:
assumes ∀ *i* < *length xs*. (∃ *y*. *xs*!*i* = *Some y*)
shows ∃ *ys*. (∀ *i* < *length xs*. *xs*!*i* = *Some* (*ys*!*i*)) ∧ *length ys* = *length xs*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *linear-term-unique-vars*:
assumes *linear-term s*
and *p* ∈ *poss s* **and** *s*|-*p* = *Var x*
and *q* ∈ *poss s* **and** *s*|-*q* = *Var x*
shows *p* = *q*
⟨*proof*⟩

lemma *linear-term-ctxt*:
assumes *linear-term t*
and *p* ∈ *poss t*
shows *vars-ctxt* (*ctxt-of-pos-term p t*) ∩ *vars-term* (*t*|-*p*) = {}
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *distinct-vars-linear-term*:
assumes *distinct* (*vars-term-list t*)
shows *linear-term t*
⟨*proof*⟩

lemma *distinct-vars-eq-linear*: *linear-term t* = *distinct* (*vars-term-list t*)
⟨*proof*⟩

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*⟩

lemma *ctxt-subst-apply*:
assumes $p \in$ *poss* t **and** $t|\text{-}p =$ *Var* x
and *linear-term* t
shows $((\textit{ctxt-of-pos-term}\ p\ t) \cdot_c \sigma)\langle s\rangle = t \cdot \sigma(x := s)$
⟨*proof*⟩

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* $p\ (t \cdot \tau)$
⟨*proof*⟩

lemma *ctxt-apply-ctxt-apply*:
assumes $p \in$ *poss* t
shows (*ctxt-of-pos-term* $(p@q)$ ((*ctxt-of-pos-term* $p\ t$) $\langle s\rangle$)) $\langle u\rangle =$ (*ctxt-of-pos-term* $p\ t$)⟨(*ctxt-of-pos-term* $q\ s$) $\langle u\rangle$⟩
⟨*proof*⟩

lemma *replace-at-append-subst*:
assumes *linear-term* t
and $p \in$ *poss* $t\ t|\text{-}p =$ *Var* x
shows (*ctxt-of-pos-term* $(p@q)\ (t \cdot \sigma))\ \langle s\rangle = t \cdot \sigma(x :=$ (*ctxt-of-pos-term* $q\ (\sigma\ x)$) $\langle s\rangle)$
⟨*proof*⟩

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
⟨*proof*⟩

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 $l \cdot \sigma = s$
⟨*proof*⟩

lemma *vars-map-vars-term*:
map f (*vars-term-list* t) = *vars-term-list* (*map-vars-term* $f\ t$)
⟨*proof*⟩

lemma *ctxt-apply-subt-at*:
assumes $q \in$ *poss* s
shows (*ctxt-of-pos-term* p $(s|\text{-}q))$ $\langle t\rangle =$ (*ctxt-of-pos-term* $(q@p)\ s$) $\langle t\rangle$ |- q
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *map2-zip*: *set* (*map fst* (*concat* (*map2 zip xs ys*))) ⊆ *set* (*concat xs*)
⟨*proof*⟩

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*⟩

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

lemma *subst-imp-mk-subst*:
assumes *s* = *t* · *σ*
shows ∃ *ss*. *t* · *σ* = *t* · (*mk-subst Var* (*zip* (*vars-distinct t*) *ss*)) ∧ *length ss* = *length* (*vars-distinct t*) ∧ (∀ *i* < *length ss*. *σ* (*vars-distinct t*!*i*) = *ss*!*i*)
⟨*proof*⟩

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*⟩

1.4 Matching Terms

The goal is showing that *match* (*t* · *σ*) *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*⟩

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*⟩

lemma *match-term-list-domain*:
assumes *match-term-list P σ = Some τ*
shows $\forall x.\ x \notin (\bigcup (a, b) \in set\ P.\ vars\text{-}term\ a) \wedge \sigma\ x = None \longrightarrow \tau\ x = None$
⟨*proof*⟩

lemma *match-subst-domain*:
assumes *match a b = Some σ*
shows *subst-domain σ ⊆ vars-term b*
⟨*proof*⟩

lemma *match-trivial*:
assumes *subst-domain σ ⊆ vars-term t*
shows *match (t · σ) t = Some σ*
⟨*proof*⟩

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* ⇒ $('v \times ('f, 'v)$ *term*$)$ *list*
where *match-substs t s = (zip (vars-term-list t) (map (λp. s|-p) (var-poss-list t)))*

lemma *mk-subst-partition-special*:
assumes *length ss = length ts*
and *is-partition (map vars-term ts)*
shows $\forall i < length\ ts.\ (ts!i) \cdot (mk\text{-}subst\ f\ (zip\ (vars\text{-}term\text{-}list\ (ts!i))\ (ss!i))) = (ts!i) \cdot (mk\text{-}subst\ f\ (concat\ (map2\ zip\ (map\ vars\text{-}term\text{-}list\ ts)\ ss)))$
⟨*proof*⟩

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*⟩

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** $\forall p \in poss\ t.\ \forall f\ ts.\ t|\text{-}p = Fun\ f\ ts \longrightarrow (\exists\ ss.\ length\ ts = length\ ss \wedge s|\text{-}p = Fun\ f\ ss)$

shows $t \cdot (\mathit{mk\text{-}subst}\ \mathit{Var}\ (\mathit{match\text{-}substs}\ t\ s)) = s$
⟨*proof*⟩

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

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

abbreviation *to-rule* :: $('f, 'v)\ \mathit{prule} \Rightarrow ('f, 'v)\ \mathit{rule}$
where *to-rule r* ≡ (*lhs r*, *rhs r*)

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

type-synonym
$('f, 'v)\ \mathit{pterm} = (('f, 'v)\ \mathit{prule} + 'f, 'v)\ \mathit{term}$
type-synonym
$('f, 'v)\ \mathit{pterm\text{-}ctxt} = (('f, 'v)\ \mathit{prule} + 'f, 'v)\ \mathit{ctxt}$

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

abbreviation *Prule* :: $('f, 'v)\ \mathit{prule} \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm}$
where *Prule* α *As* ≡ *Fun* (*Inl* α) *As*
abbreviation *Pfun* :: $'f \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm}$
where *Pfun f As* ≡ *Fun* (*Inr f*) *As*

Also for contexts.

abbreviation *Crule* :: $('f, 'v)\ \mathit{prule} \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm\text{-}ctxt} \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm\text{-}ctxt}$
where *Crule* α *As1 C As2* ≡ *More* (*Inl* α) *As1 C As2*
abbreviation *Cfun* :: $'f \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm\text{-}ctxt} \Rightarrow ('f, 'v)\ \mathit{pterm\ list} \Rightarrow ('f, 'v)\ \mathit{pterm\text{-}ctxt}$
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*]:

$(\bigwedge x.\ A = Var\ x \Longrightarrow P) \Longrightarrow (\bigwedge f\ As.\ A = Pfun\ f\ As \Longrightarrow P) \Longrightarrow (\bigwedge \alpha\ As.\ A = Prule\ \alpha\ As \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

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 $\bigwedge \alpha\ As.\ (\bigwedge a.\ a \in set\ As \Longrightarrow P\ a) \Longrightarrow P\ (Prule\ \alpha\ As)$
shows *pterm-induct* [*case-names Var Pfun Prule*, *induct type*: *pterm*]: $P\ A$
⟨*proof*⟩

Induction scheme for contexts of proof terms.

lemma
fixes $P :: ('f, 'v)\ pterm\text{-}ctxt \Rightarrow bool$
assumes $P\ \Box$
and $\bigwedge f\ ss1\ C\ ss2.\ P\ C \Longrightarrow P\ (Cfun\ f\ ss1\ C\ ss2)$
and $\bigwedge \alpha\ ss1\ C\ ss2.\ P\ C \Longrightarrow P\ (Crule\ \alpha\ ss1\ C\ ss2)$
shows *pterm-ctxt-induct* [*case-names Hole Cfun Crule*, *induct type*: *pterm-ctxt*]: $P\ C$
⟨*proof*⟩

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** $var\text{-}rule\ \alpha \equiv vars\text{-}distinct\ (lhs\ \alpha)$

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

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

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

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

Source and target

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

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* ($\lambda(l, r)$. *is-Fun l*)

locale *var-rhs-subset-lhs* =
  fixes *R* :: $('f, 'v)$ *trs*
  assumes *varcond*:*Ball R* ($\lambda(l, r)$. *vars-term r* $\subseteq$ *vars-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*
⟨*proof*⟩
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))
⟨*proof*⟩
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* ⇒ ('*f*, '*v*) *pterm set*
for *R* **where**
[*simp*]: *Var x* ∈ *wf-pterm R*
|[*intro*]: ∀ *t* ∈ *set ts*. *t* ∈ *wf-pterm R* ⟹ *Pfun f ts* ∈ *wf-pterm R*
|[*intro*]: (*lhs* α, *rhs* α) ∈ *R* ⟹ *length As* = *length* (*var-rule* α) ⟹
∀ *a* ∈ *set As*. *a* ∈ *wf-pterm R* ⟹ *Prule* α *As* ∈ *wf-pterm R*

inductive-set *wf-pterm-ctxt* :: ('*f*, '*v*) *trs* ⇒ ('*f*, '*v*) *pterm-ctxt set*
for *R* **where**
[*simp*]: □ ∈ *wf-pterm-ctxt R*
|[*intro*]: ∀ *s* ∈ (*set ss1*) ∪ (*set ss2*). *s* ∈ *wf-pterm R* ⟹ *C* ∈ *wf-pterm-ctxt R* ⟹
Cfun f ss1 C ss2 ∈ *wf-pterm-ctxt R*
|[*intro*]: (*lhs* α, *rhs* α) ∈ *R* ⟹ (*length ss1*) + (*length ss2*) + 1 = *length* (*var-rule* α) ⟹
∀ *s* ∈ (*set ss1*) ∪ (*set ss2*). *s* ∈ *wf-pterm R* ⟹ *C* ∈ *wf-pterm-ctxt R* ⟹
Crule α *ss1 C ss2* ∈ *wf-pterm-ctxt R*

lemma *fun-well-arg*[*intro*]:
assumes *Fun f As* ∈ *wf-pterm R a* ∈ *set As*
shows *a* ∈ *wf-pterm R*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *trs-well-ctxt-C*[*intro*]:
assumes *More f ss1 C ss2* ∈ *wf-pterm-ctxt R*
shows *C* ∈ *wf-pterm-ctxt R*
⟨*proof*⟩

context *no-var-lhs*
begin
lemma *lhs-is-Fun*:
assumes *Prule* α *Bs* ∈ *wf-pterm R*
shows *is-Fun* (*lhs* α)
⟨*proof*⟩
end

lemma *lhs-subst-var-well-def*:
assumes ∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R*
shows (⟨*As*⟩$_\alpha$) *x* ∈ *wf-pterm R*
⟨*proof*⟩

lemma *lhs-subst-well-def*:
assumes ∀ *i* < *length As*. *As*!*i* ∈ *wf-pterm R B* ∈ *wf-pterm R*
shows *B* · (⟨*As*⟩$_\alpha$) ∈ *wf-pterm R*
⟨*proof*⟩

lemma *subt-at-is-wf-pterm*:
assumes *p* ∈ *poss A* **and** *A* ∈ *wf-pterm R*
shows *A*|-*p* ∈ *wf-pterm R*
⟨*proof*⟩

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*
⟨*proof*⟩

Every normal term is a well-defined proof term.

lemma *to-pterm-wf-pterm*[*simp*]: *to-pterm t* ∈ *wf-pterm R*
⟨*proof*⟩

lemma *to-pterm-trs-ctxt*:
assumes *p* ∈ *poss* (*to-pterm s*)
shows *ctxt-of-pos-term p* (*to-pterm s*) ∈ *wf-pterm-ctxt R*
⟨*proof*⟩

lemma *to-pterm-ctxt-wf-pterm-ctxt*:
shows *to-pterm-ctxt C* ∈ *wf-pterm-ctxt R*
⟨*proof*⟩

lemma *ctxt-wf-pterm*:
assumes *A* ∈ *wf-pterm R* **and** *p* ∈ *poss A*
and *B* ∈ *wf-pterm R*

shows $(ctxt\text{-}of\text{-}pos\text{-}term\ p\ A)\langle B\rangle \in wf\text{-}pterm\ R$
⟨*proof*⟩

2.4 'Normal' Terms vs. Proof Terms

lemma *to-pterm-empty*: $is\text{-}empty\text{-}step\ (to\text{-}pterm\ t)$
⟨*proof*⟩

Variables remain unchanged.

lemma *vars-to-pterm*: $vars\text{-}term\text{-}list\ (to\text{-}pterm\ t) = vars\text{-}term\text{-}list\ t$
⟨*proof*⟩

lemma *poss-list-to-pterm*: $poss\text{-}list\ t = poss\text{-}list\ (to\text{-}pterm\ t)$
⟨*proof*⟩

lemma *p-in-poss-to-pterm*:
assumes $p \in poss\ t$
shows $p \in poss\ (to\text{-}pterm\ t)$
⟨*proof*⟩

lemma *var-poss-to-pterm*: $var\text{-}poss\ t = var\text{-}poss\ (to\text{-}pterm\ t)$
⟨*proof*⟩

lemma *var-poss-list-to-pterm*: $var\text{-}poss\text{-}list\ (to\text{-}pterm\ t) = var\text{-}poss\text{-}list\ t$
⟨*proof*⟩

to-pterm distributes over application of substitution.

lemma *to-pterm-subst*:
$to\text{-}pterm\ (t \cdot \sigma) = (to\text{-}pterm\ t) \cdot (to\text{-}pterm \circ \sigma)$
⟨*proof*⟩

to-pterm distributes over context.

lemma *to-pterm-ctxt-of-pos-apply-term*:
assumes $p \in poss\ s$
shows $to\text{-}pterm\ ((ctxt\text{-}of\text{-}pos\text{-}term\ p\ s)\ \langle t\rangle) = (ctxt\text{-}of\text{-}pos\text{-}term\ p\ (to\text{-}pterm\ s))\langle to\text{-}pterm\ t\rangle$
⟨*proof*⟩

Linear terms become linear proof terms.

lemma *to-pterm-linear*:
assumes *linear-term t*
shows *linear-term* (*to-pterm t*)
⟨*proof*⟩

lemma *lhs-subst-trivial*:
shows $match\ (to\text{-}pterm\ (lhs\ \alpha) \cdot \langle As\rangle_\alpha)\ (to\text{-}pterm\ (lhs\ \alpha)) = Some\ \langle As\rangle_\alpha$
⟨*proof*⟩

lemma *to-pterm-ctxt-apply-term*:

to-pterm $C\langle t\rangle$ = (*to-pterm-ctxt* C) $\langle$*to-pterm* $t\rangle$
⟨*proof*⟩

2.5 Substitutions

lemma *fun-mk-subst*[*simp*]:
assumes $\forall x.\ f\ (Var\ x) = Var\ x$
shows $f \circ (mk\text{-}subst\ Var\ (zip\ vs\ ts)) = mk\text{-}subst\ Var\ (zip\ vs\ (map\ f\ ts))$
⟨*proof*⟩

lemma *apply-lhs-subst-var-rule*:
assumes *length ts* = *length* (*var-rule* α)
shows *map* ($\langle ts\rangle_\alpha$) (*var-rule* α) = *ts*
⟨*proof*⟩

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-pterm* (*lhs* α)) $\cdot$ $\langle Bs\rangle_\alpha$ $\wedge$
$(\forall x \in set\ (var\text{-}rule\ \alpha).\ \sigma\ x = (\langle Bs\rangle_\alpha)\ x)$
⟨*proof*⟩

lemma *apply-subst-wf-pterm*:
assumes $A \in$ *wf-pterm R*
and $\forall x \in vars\text{-}term\ A.\ \sigma\ x \in wf\text{-}pterm\ R$
shows $A \cdot \sigma \in$ *wf-pterm R*
⟨*proof*⟩

lemma *subst-well-def*:
assumes $B \in$ *wf-pterm R* $A \cdot \sigma = B\ x \in vars\text{-}term\ A$
shows $\sigma\ x \in$ *wf-pterm R*
⟨*proof*⟩

lemma *lhs-subst-args-wf-pterm*:
assumes *to-pterm* (*lhs* α) $\cdot$ $\langle As\rangle_\alpha \in$ *wf-pterm R* **and** *length As* = *length* (*var-rule* α)
shows $\forall a \in set\ As.\ a \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *match-well-def*:
assumes $B \in$ *wf-pterm R match B A* = *Some* σ
shows $\forall i < length\ (vars\text{-}distinct\ A).\ \sigma\ ((vars\text{-}distinct\ A)\ !\ i) \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *subst-imp-well-def*:
assumes $A \cdot \sigma \in$ *wf-pterm R*
shows $A \in$ *wf-pterm R*
⟨*proof*⟩

lemma *lhs-subst-var-i*:
assumes $x = (\mathit{var\text{-}rule}\ \alpha)!i$ **and** $i < \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$ **and** $i < \mathit{length}\ \mathit{As}$
shows $(\langle \mathit{As} \rangle_\alpha)\ x = \mathit{As}!i$
⟨*proof*⟩

lemma *lhs-subst-not-var-i*:
assumes $\neg(\exists\, i < \mathit{length}\ \mathit{As}.\ i < \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha) \wedge x = (\mathit{var\text{-}rule}\ \alpha)!i)$
shows $(\langle \mathit{As} \rangle_\alpha)\ x = \mathit{Var}\ x$
⟨*proof*⟩

lemma *lhs-subst-upd*:
assumes $\mathit{length}\ \mathit{ss1} < \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$
shows $((\langle \mathit{ss1} \mathbin{@} t \# \mathit{ss2} \rangle_\alpha)\ ((\mathit{var\text{-}rule}\ \alpha)!(\mathit{length}\ \mathit{ss1}) := s)) = \langle \mathit{ss1} \mathbin{@} s \# \mathit{ss2} \rangle_\alpha$
⟨*proof*⟩

lemma *eval-lhs-subst*:
assumes l:$\mathit{length}\ (\mathit{var\text{-}rule}\ \alpha) = \mathit{length}\ \mathit{As}$
shows $(\mathit{to\text{-}pterm}\ (\mathit{lhs}\ \alpha)) \cdot \langle \mathit{As} \rangle_\alpha \cdot \sigma = (\mathit{to\text{-}pterm}\ (\mathit{lhs}\ \alpha)) \cdot \langle \mathit{map}\ (\lambda a.\ a \cdot \sigma)\ \mathit{As} \rangle_\alpha$
⟨*proof*⟩

lemma *var-rule-pos-subst*:
assumes $i < \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$ $\mathit{length}\ \mathit{ss} = \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$
and $p \in \mathit{poss}\ (\mathit{lhs}\ \alpha)$ $\mathit{Var}\ ((\mathit{var\text{-}rule}\ \alpha)!i) = (\mathit{lhs}\ \alpha)|\text{-}p$
shows $\mathit{lhs}\ \alpha \cdot \langle \mathit{ss} \rangle_\alpha\ |\text{-}\ (p@q) = (\mathit{ss}!i)|\text{-}q$
⟨*proof*⟩

lemma *lhs-subst-var-rule*:
assumes $\mathit{vars\text{-}term}\ t \subseteq \mathit{vars\text{-}term}\ (\mathit{lhs}\ \alpha)$
shows $t \cdot \langle \mathit{map}\ \sigma\ (\mathit{var\text{-}rule}\ \alpha) \rangle_\alpha = t \cdot \sigma$
⟨*proof*⟩

2.6 Contexts

lemma *match-lhs-context*:
assumes $i < \mathit{length}\ (\mathit{vars\text{-}term\text{-}list}\ t) \wedge p = (\mathit{var\text{-}poss\text{-}list}\ t)!i$
and $\mathit{linear\text{-}term}\ t$
and $\mathit{match}\ (((\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ p\ (t \cdot \sigma)))\langle B \rangle)\ t = \mathit{Some}\ \tau$
shows $\mathit{map}\ \tau\ (\mathit{vars\text{-}term\text{-}list}\ t) = (\mathit{map}\ \sigma\ (\mathit{vars\text{-}term\text{-}list}\ t))[i := B]$
⟨*proof*⟩

lemma *ctxt-lhs-subst*:
assumes i:$i < \mathit{length}\ (\mathit{var\text{-}poss\text{-}list}\ (\mathit{lhs}\ \alpha))$ **and** l:$\mathit{length}\ \mathit{As} = \mathit{length}\ (\mathit{var\text{-}rule}\ \alpha)$
and $p1$:$p1 = \mathit{var\text{-}poss\text{-}list}\ (\mathit{lhs}\ \alpha)\ !\ i$ **and** lin:$\mathit{linear\text{-}term}\ (\mathit{lhs}\ \alpha)$
and $p2 \in \mathit{poss}\ (\mathit{As}!i)$
shows $(\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ (p1 \mathbin{@} p2)\ (\mathit{to\text{-}pterm}\ (\mathit{lhs}\ \alpha) \cdot \langle \mathit{As} \rangle_\alpha))\langle A \rangle =$
$(\mathit{to\text{-}pterm}\ (\mathit{lhs}\ \alpha)) \cdot \langle \mathit{take}\ i\ \mathit{As} \mathbin{@} (\mathit{ctxt\text{-}of\text{-}pos\text{-}term}\ p2\ (\mathit{As}!i))\langle A \rangle \# \mathit{drop}\ (\mathit{Suc}\ i)\ \mathit{As} \rangle_\alpha$

⟨*proof*⟩

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*⟩

2.7 Source and Target

lemma *source-empty-step*:
assumes *is-empty-step t*
shows *to-pterm* (*source t*) = *t*
⟨*proof*⟩

lemma *empty-coinitial*:
shows *co-initial A t* ⟹ *is-empty-step t* ⟹ *to-pterm* (*source A*) = *t*
⟨*proof*⟩

lemma *source-to-pterm*[*simp*]: *source* (*to-pterm t*) = *t*
⟨*proof*⟩

lemma *target-to-pterm*[*simp*]: *target* (*to-pterm t*) = *t*
⟨*proof*⟩

lemma *vars-term-source*:
assumes *A* ∈ *wf-pterm R*
shows *vars-term A* = *vars-term* (*source A*)
⟨*proof*⟩

context *var-rhs-subset-lhs*
begin
lemma *vars-term-target*:
assumes *A* ∈ *wf-pterm R*
shows *vars-term* (*target A*) ⊆ *vars-term A*
⟨*proof*⟩
end

lemma *linear-source-imp-linear-pterm*:
assumes *A* ∈ *wf-pterm R linear-term* (*source A*)
shows *linear-term A*
⟨*proof*⟩

context *var-rhs-subset-lhs*

begin
lemma *target-apply-subst*:
  assumes $A \in$ *wf-pterm* R
  shows *target* $(A \cdot \sigma) =$ (*target* A) $\cdot$ (*target* $\circ\ \sigma$)
⟨*proof*⟩
end

context *var-rhs-subset-lhs*
begin
lemma *tgt-subst-simp*:
assumes $A \in$ *wf-pterm* R
  shows *target* $(A \cdot \sigma) =$ *target* ((*to-pterm* (*target* A)) $\cdot\ \sigma$)
  ⟨*proof*⟩
end

lemma *target-empty-apply-subst*:
  assumes *is-empty-step* t
  shows *target* $(t \cdot \sigma) =$ (*target* t) $\cdot$ (*target* $\circ\ \sigma$)
⟨*proof*⟩

lemma *source-ctxt-comp*:*source-ctxt* (*C1* $\circ_c$ *C2*) = *source-ctxt C1* $\circ_c$ *source-ctxt C2*
  ⟨*proof*⟩

lemma *context-source*: *source* $(A\langle B\rangle)$ = *source* $(A\langle$*to-pterm* (*source* $B)\rangle)$
⟨*proof*⟩

lemma *context-target*: *target* $(A\langle B\rangle)$ = *target* $(A\langle$*to-pterm* (*target* $B)\rangle)$
⟨*proof*⟩

lemma *source-to-pterm-ctxt*:
  source $(($*to-pterm-ctxt* $C)\langle A\rangle) = C\langle$*source* $A\rangle$
  ⟨*proof*⟩

lemma *target-to-pterm-ctxt*:
  target $(($*to-pterm-ctxt* $C)\langle A\rangle) = C\langle$*target* $A\rangle$
  ⟨*proof*⟩

lemma *source-ctxt-to-pterm*:
  assumes $p \in$ *poss* s
  shows *source-ctxt* (*ctxt-of-pos-term* p (*to-pterm* s)) = *ctxt-of-pos-term* p s
⟨*proof*⟩

lemma *to-pterm-ctxt-at-pos*:
  assumes $p \in$ *poss* s
  shows *ctxt-of-pos-term* p (*to-pterm* s) = *to-pterm-ctxt* (*ctxt-of-pos-term* p s)
⟨*proof*⟩

lemma *to-pterm-ctxt-hole-pos*: *hole-pos* C = *hole-pos* (*to-pterm-ctxt* C)

⟨*proof*⟩

lemma *source-to-pterm-ctxt'*:
assumes $q \in$ *poss* s
shows *source-ctxt* (*to-pterm-ctxt* (*ctxt-of-pos-term* q s)) = *ctxt-of-pos-term* q s
⟨*proof*⟩

lemma *to-pterm-ctxt-comp*: *to-pterm-ctxt* ($C \circ_c D$) = *to-pterm-ctxt* $C \circ_c$ *to-pterm-ctxt* D
⟨*proof*⟩

lemma *source-apply-subst*:
assumes $A \in$ *wf-pterm* R
shows *source* ($A \cdot \sigma$) = (*source* A) $\cdot$ (*source* $\circ$ σ)
⟨*proof*⟩

lemma *ctxt-of-pos-term-at-var-subst*:
assumes *linear-term* t
and $p \in$ *poss* t **and** $t|_p$ = *Var* x
and $\forall y \in$ *vars-term* t. $y \neq x \longrightarrow \tau\ y = \sigma\ y$
shows *ctxt-of-pos-term* p ($t \cdot \tau$) = *ctxt-of-pos-term* p ($t \cdot \sigma$)
⟨*proof*⟩

context *left-lin*
begin

lemma *source-ctxt-apply-subst*:
assumes $C \in$ *wf-pterm-ctxt* R
shows *source-ctxt* ($C \cdot_c \sigma$) = (*source-ctxt* C) $\cdot_c$ (*source* $\circ$ σ)
⟨*proof*⟩

Needs left-linearity to avoid multihole contexts.

lemma *source-ctxt-apply-term*:
assumes $C \in$ *wf-pterm-ctxt* R
shows *source* ($C\langle A\rangle$) = (*source-ctxt* C)⟨*source* A⟩
⟨*proof*⟩
end

lemma *rewrite-tgt*:
assumes *rstep*:$(t,v) \in$ (*rstep* R)*
shows (*target* (C ⟨(*to-pterm* t) $\cdot$ σ⟩), *target* (C ⟨(*to-pterm* v) $\cdot$ σ⟩)) $\in$ (*rstep* R)*
⟨*proof*⟩

2.8 Additional Results

lemma *length-args-well-Prule*:
assumes *Prule* α *As* $\in$ *wf-pterm* R *Prule* α *Bs* $\in$ *wf-pterm* S
shows *length As* = *length Bs*
⟨*proof*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

2.9 Proof Terms Represent Multi-Steps

context *var-rhs-subset-lhs*
begin
lemma *mstep-to-pterm*:
assumes $(s, t) \in mstep\ R$
shows $\exists A.\ A \in wf\text{-}pterm\ R \wedge source\ A = s \wedge target\ A = t$
⟨*proof*⟩
end

lemma *pterm-to-mstep*:
assumes $A \in wf\text{-}pterm\ R$
shows $(source\ A, target\ A) \in mstep\ R$
⟨*proof*⟩

lemma *co-init-prule*:
assumes *co-initial* (*Prule α As*) (*Prule α Bs*)
and *Prule α As* ∈ *wf-pterm R* **and** *Prule α Bs* ∈ *wf-pterm R*
shows ∀ *i*<*length As*. *co-initial* (*As*!*i*) (*Bs*!*i*)
⟨*proof*⟩

3 Operations on Proof Terms

The operations residual, deletion, and join on proof terms all fulfill $A \star (source\ A) = A$ 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*: $A \in$ *wf-pterm* $R \Longrightarrow f\ A$ (*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 $C \in$ *wf-pterm-ctxt* R
and $A \star B$ = *Some* D
shows $C\langle A\rangle \star$ (*to-pterm-ctxt* (*source-ctxt* C))$\langle B\rangle$ = *Some* ($C\langle D\rangle$)
⟨*proof*⟩

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 $x \in$ *vars-term t*
shows *size* (σ x) ≤ *size s*
⟨*proof*⟩

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*⟩

Additional simp rule because we allow variable left-hand sides of rewrite rules at this point. Then $Var\ x\ /\ \alpha$ and $\alpha\ /\ Var\ x$ 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*⟩

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*⟩$_\alpha$)
      | *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*⟩$_\alpha$)
      | *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 $f = g \wedge$ *length As* $=$ *length Bs* $\wedge$
$(\exists\, Cs.\ C =$ *Pfun f Cs* $\wedge$
length Cs $=$ *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*$!i$ *re Bs*$!i =$ *Some* (*Cs*$!i$)))
⟨*proof*⟩

lemma *residual-rule-rule*:
assumes (*Prule* α *As*) *re* (*Prule* β *Bs*) $=$ *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
(*Prule* β *Bs*) $\in$ *wf-pterm S*
shows $\alpha = \beta \wedge$ *length As* $=$ *length Bs* $\wedge$
$(\exists\, Cs.\ C =$ *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*$!i$)))
⟨*proof*⟩

lemma *residual-rule-var*:
assumes (*Prule* α *As*) *re* (*Var x*) $=$ *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
shows $\exists\sigma$. *match* (*Var x*) (*to-pterm* (*lhs* α)) $=$ *Some* $\sigma \wedge$
$(\exists\, Cs.\ C =$ *Prule* α *Cs* $\wedge$
length Cs $=$ *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*$!i$ *re* (σ (*var-rule* α ! i)) $=$ *Some* (*Cs*$!i$)))
⟨*proof*⟩

lemma *residual-rule-fun*:
assumes (*Prule* α *As*) *re* (*Pfun f Bs*) $=$ *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
shows $\exists\sigma$. *match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) $=$ *Some* $\sigma \wedge$
$(\exists\, Cs.\ C =$ *Prule* α *Cs* $\wedge$
length Cs $=$ *length As* $\wedge$
$(\forall\, i <$ *length As*. *As*$!i$ *re* (σ (*var-rule* α ! i)) $=$ *Some* (*Cs*$!i$)))
⟨*proof*⟩

lemma *residual-var-rule*:
assumes (*Var x*) *re* (*Prule* α *As*) $=$ *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
shows $\exists\sigma$. *match* (*Var x*) (*to-pterm* (*lhs* α)) $=$ *Some* $\sigma \wedge$
$(\exists\, Cs.\ C =$ (*to-pterm* (*rhs* α)) $\cdot \langle Cs \rangle_\alpha \wedge$
length Cs $=$ *length As* $\wedge$
$(\forall\, i <$ *length As*. (σ (*var-rule* α ! i) *re As*$!i$) $=$ *Some* (*Cs*$!i$)))
⟨*proof*⟩

lemma *residual-fun-rule*:
assumes (*Pfun f Bs*) *re* (*Prule* α *As*) $=$ *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
shows $\exists\sigma$. *match* (*Pfun f Bs*) (*to-pterm* (*lhs* α)) $=$ *Some* $\sigma \wedge$
$(\exists\, Cs.\ C =$ (*to-pterm* (*rhs* α)) $\cdot \langle Cs \rangle_\alpha \wedge$
length Cs $=$ *length As* $\wedge$

$(\forall\, i < \mathit{length}\ \mathit{As}.\ (\sigma\ (\mathit{var\text{-}rule}\ \alpha\ !\ i))\ \mathit{re}\ \mathit{As}!i = \mathit{Some}\ (\mathit{Cs}!i)))$
⟨*proof*⟩

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

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*⟩

$A\ /\ t = A$

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

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

lemma *res-same*: *A re A* = *Some* (*to-pterm* (*target A*))
⟨*proof*⟩

lemma *residual-src-tgt*:
assumes *A re B* = *Some C A* ∈ *wf-pterm R B* ∈ *wf-pterm S*
shows *source C* = *target B*
⟨*proof*⟩

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 ⋀σ *a b c d. 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 R* ⟹ *b* ∈ *wf-pterm S* ⟹
target c = *target d*
Prule α *As re Var v* = *Some C*
Var v re Prule α *As* = *Some D*
Prule α *As* ∈ *wf-pterm R*
shows *target C* = *target D*
⟨*proof*⟩

lemma *tgt-tgt-rule-fun*:
assumes ⋀σ *a b c d. 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 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*⟩

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*
⟨*proof*⟩

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*⟩

lemma *residual-well-defined*:
assumes *A* ∈ *wf-pterm R B* ∈ *wf-pterm S A re B* = *Some C*
shows *C* ∈ *wf-pterm R*
⟨*proof*⟩

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*: $A \sqcup B = B \sqcup A$
⟨*proof*⟩

lemma *join-with-source*:
assumes $A \in$ *wf-pterm R*
shows $A \sqcup$ *to-pterm* (*source A*) = *Some A*
⟨*proof*⟩

context *no-var-lhs*
begin

lemma *join-subst*:
assumes $B \in$ *wf-pterm R* **and** $\forall x \in$ *vars-term B*. $\varrho\ x \in$ *wf-pterm R*
and $\forall x \in$ *vars-term B*. *source* ($\varrho\ x$) = $\sigma\ x$
shows ($B \cdot$ (*to-pterm* $\circ\ \sigma$)) $\sqcup$ ((*to-pterm* (*source B*)) $\cdot\ \varrho$) = *Some* ($B \cdot \varrho$)
⟨*proof*⟩

end

lemma *join-same*:
shows $A \sqcup A$ = *Some A*
⟨*proof*⟩

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*) $\sqcup$ (*Pfun g Bs*) = *Some C*
shows $f = g \wedge$ *length As* = *length Bs* $\wedge$
($\exists$ *Cs*. *C* = *Pfun f Cs* $\wedge$
length Cs = *length As* $\wedge$
($\forall i <$ *length As*. *As*!$i \sqcup$ *Bs*!i = *Some* (*Cs*!i)))
⟨*proof*⟩

lemma *join-rule-rule*:
assumes (*Prule* α *As*) $\sqcup$ (*Prule* β *Bs*) = *Some C*
(*Prule* α *As*) $\in$ *wf-pterm R*
(*Prule* β *Bs*) $\in$ *wf-pterm R*
shows $\alpha = \beta \wedge$ *length As* = *length Bs* $\wedge$
($\exists$ *Cs*. *C* = *Prule* α *Cs* $\wedge$
length Cs = *length As* $\wedge$
($\forall i <$ *length As*. *As*!$i \sqcup$ *Bs*!i = *Some* (*Cs*!i)))
⟨*proof*⟩

lemma *join-rule-var*:
assumes $(Prule\ \alpha\ As) \sqcup (Var\ x) = Some\ C$
$(Prule\ \alpha\ As) \in wf\text{-}pterm\ R$
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*⟩

lemma *join-rule-fun*:
assumes $(Prule\ \alpha\ As) \sqcup (Pfun\ f\ Bs) = Some\ C$
$(Prule\ \alpha\ As) \in wf\text{-}pterm\ R$
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*⟩

lemma *join-wf-pterm*:
assumes $A \sqcup B = Some\ C$
and $A \in wf\text{-}pterm\ R$ **and** $B \in wf\text{-}pterm\ R$
shows $C \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *source-join*:
assumes $A \sqcup B = Some\ C$
and $A \in wf\text{-}pterm\ R$ **and** $B \in wf\text{-}pterm\ R$
shows *co-initial A C*
⟨*proof*⟩

lemma *join-pterm-subst-Some*:
fixes $A\ B{::}('f, 'v)\ pterm$
assumes $(A \cdot \sigma) \sqcup (A \cdot \tau) = Some\ B$
shows $\exists \varrho.\ (\forall\, x \in vars\text{-}term\ A.\ \sigma\ x \sqcup \tau\ x = Some\ (\varrho\ x)) \wedge B = A \cdot \varrho \wedge match\ B\ A = Some\ \varrho$
⟨*proof*⟩

lemma *join-pterm-subst-None*:
fixes $A{::}('f, 'v)\ pterm$
assumes $(A \cdot \sigma) \sqcup (A \cdot \tau) = None$
shows $\exists\ x \in vars\text{-}term\ A.\ \sigma\ x \sqcup \tau\ x = None$
⟨*proof*⟩

fun *mk-subst-from-list* :: $('v \Rightarrow ('f, 'v)\ term)\ list \Rightarrow ('v \Rightarrow ('f, 'v)\ term)$ **where**
mk-subst-from-list $[] = Var$
| *mk-subst-from-list* $(\sigma \#\ \sigma s) = (\lambda x.\ case\ \sigma\ x\ of$
$Var\ x \Rightarrow mk\text{-}subst\text{-}from\text{-}list\ \sigma s\ x$
$\mid t \Rightarrow t)$

lemma *join-is-Fun*:
  assumes *join*:$A \sqcup B = Some\ (Pfun\ f\ Cs)$
  shows $\exists As.\ A = Pfun\ f\ As \wedge length\ As = length\ Cs$
⟨*proof*⟩

lemma *join-obtain-subst*:
  assumes *join*:$A \sqcup B = Some\ (to\text{-}pterm\ t \cdot \sigma)$ **and** *linear-term t*
  shows $(to\text{-}pterm\ t) \cdot mk\text{-}subst\ Var\ (match\text{-}substs\ (to\text{-}pterm\ t)\ A) = A$
⟨*proof*⟩

lemma *join-pterm-linear-subst*:
  assumes *join*:$A \sqcup B = Some\ (to\text{-}pterm\ t \cdot \sigma)$ **and** *lin*:*linear-term t*
  shows $\exists\ \sigma_A\ \sigma_B.\ A = (to\text{-}pterm\ t \cdot \sigma_A) \wedge B = (to\text{-}pterm\ t \cdot \sigma_B) \wedge (\forall x \in vars\text{-}term\ t.\ \sigma_A\ x \sqcup \sigma_B\ x = Some\ (\sigma\ x))$
⟨*proof*⟩

context *no-var-lhs*
begin
lemma *join-rule-lhs*:
  assumes *wf*:$Prule\ \alpha\ As \in wf\text{-}pterm\ R$ **and** *args*:$\forall i < length\ As.\ As!i \sqcup Bs!i \neq None$ **and** *l*:$length\ Bs = length\ As$
  shows $Prule\ \alpha\ As \sqcup (to\text{-}pterm\ (lhs\ \alpha) \cdot \langle Bs \rangle_\alpha) \neq None$
⟨*proof*⟩
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
    $join\text{-}opt\ A\ (Some\ B) = A \sqcup B$
  | $join\text{-}opt\ \text{-}\ \text{-} = None$

fun *join-list* :: $('f, 'v)\ pterm\ list \Rightarrow ('f,'v)\ pterm\ option$ $(\bigsqcup)$
  where
    $join\text{-}list\ [] = None$
  | $join\text{-}list\ (A\ \#\ []) = Some\ A$
  | $join\text{-}list\ (A\ \#\ As) = join\text{-}opt\ A\ (join\text{-}list\ As)$

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

lemma *join-var-rule*:
  assumes *to-rule* $\alpha \in R$
  shows $Var\ x \sqcup Prule\ \alpha\ As = None$
⟨*proof*⟩

lemma *var-join*:
assumes $Var\ x \sqcup B = Some\ C$ **and** $B \in wf\text{-}pterm\ R$
shows $B = Var\ x \wedge C = Var\ x$
⟨*proof*⟩

lemma *fun-join*:
assumes $Pfun\ f\ As \sqcup B = Some\ C$
shows $(\exists\, g\ Bs.\ B = Pfun\ g\ Bs) \vee (\exists\, \alpha\ Bs.\ B = Prule\ \alpha\ Bs)$
⟨*proof*⟩

lemma *rule-join*:
assumes $Prule\ \alpha\ As \sqcup B = Some\ C$ **and** $Prule\ \alpha\ As \in wf\text{-}pterm\ R$
shows $(\exists\, g\ Bs.\ B = Pfun\ g\ Bs) \vee (\exists\, \beta\ Bs.\ B = Prule\ \beta\ Bs)$
⟨*proof*⟩

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 $A \in wf\text{-}pterm\ R\ B \in wf\text{-}pterm\ R\ C \in wf\text{-}pterm\ R$
shows $join\text{-}opt\ A\ (B \sqcup C) = join\text{-}opt\ C\ (A \sqcup B)$
⟨*proof*⟩

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

lemma *join-triple-defined*:
assumes $A \in wf\text{-}pterm\ R\ B \in wf\text{-}pterm\ R\ C \in wf\text{-}pterm\ R$
and $A \sqcup B \neq None\ B \sqcup C \neq None\ A \sqcup C \neq None$
shows $join\text{-}opt\ A\ (B \sqcup C) \neq None$
⟨*proof*⟩

lemma *join-list-defined*:
assumes $\forall\ a1\ a2.\ a1 \in set\ As \wedge a2 \in set\ As \longrightarrow a1 \sqcup a2 \neq None$
and $\forall\, a \in set\ As.\ a \in wf\text{-}pterm\ R$ **and** $As \neq []$
shows $\exists\ D.\ join\text{-}list\ As = Some\ D \wedge D \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *join-list-wf-pterm*:
assumes $\forall\, a \in set\ As.\ a \in wf\text{-}pterm\ R$
and $join\text{-}list\ As = Some\ B$
shows $B \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *source-join-list*:
assumes $join\text{-}list\ As = Some\ B$ **and** $\forall\, a \in set\ As.\ a \in wf\text{-}pterm\ R$
shows $\bigwedge A.\ A \in set\ As \Longrightarrow source\ A = source\ B$
⟨*proof*⟩

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*
⟨*proof*⟩

context *no-var-lhs*
begin
lemma *deletion-source*:
assumes *A* ∈ *wf-pterm R B* ∈ *wf-pterm R*
and *A* $-_p$ *B* = *Some C*
shows *source C* = *source A*
⟨*proof*⟩
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* ∘ (λ*pi*. *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 $p \in$ *poss* s
shows *source* (*ll-single-redex* s p α) = (*ctxt-of-pos-term* p s)⟨(*lhs* α) · ⟨*map* (λ*pi*. s|-(p@*pi*)) (*var-poss-list* (*lhs* α))⟩$_\alpha$⟩
⟨*proof*⟩

lemma *target-single-redex*:
assumes $p \in$ *poss* s
shows *target* (*ll-single-redex* s p α) = (*ctxt-of-pos-term* p s)⟨(*rhs* α) · ⟨*map* (λ*pi*. s|-(p@*pi*)) (*var-poss-list* (*lhs* α))⟩$_\alpha$⟩
⟨*proof*⟩

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)) ∈ *rstep* R
⟨*proof*⟩

lemma *single-redex-neq*:
assumes (α, p) ≠ (β, q) $p \in$ *poss* s $q \in$ *poss* s
shows *ll-single-redex* s p α ≠ *ll-single-redex* s q β
⟨*proof*⟩

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* $\alpha \in R$ ∧ $p \in$ *poss* s
⟨*proof*⟩
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*⟩

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*:$A \in$ *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*
⟨*proof*⟩

interpretation *deletion-op*:*op-proof-term R deletion*
⟨*proof*⟩

abbreviation $As \equiv (map\ (\lambda i.\ A|\text{-}(q@[i]))\ [0..<length\ (var\text{-}rule\ \alpha)])$

lemma *length-as*:$length\ As = length\ (var\text{-}rule\ \alpha)$
⟨*proof*⟩

lemma *as-well*:$\forall\, i < length\ As.\ As!i \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *a*:$A = (ctxt\text{-}of\text{-}pos\text{-}term\ q\ A)\langle Prule\ \alpha\ As\rangle$
⟨*proof*⟩

lemma *rule-in-TRS*: $to\text{-}rule\ \alpha \in R$
⟨*proof*⟩

lemma *lin-lhs*:$linear\text{-}term\ (lhs\ \alpha)$
⟨*proof*⟩

lemma *source-at-pq*:$source\ (A|\text{-}q) = (source\ A)|\text{-}p$
⟨*proof*⟩

lemma *single-redex-pterm*:
shows $\Delta = (ctxt\text{-}of\text{-}pos\text{-}term\ p\ (to\text{-}pterm\ (source\ A)))\langle Prule\ \alpha\ (map\ (to\text{-}pterm \circ source)\ As)\rangle$
⟨*proof*⟩

lemma *delta-trs-wf-pterm*:
shows $\Delta \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *source-delta*: $source\ \Delta = source\ A$
⟨*proof*⟩

lemma *residual*:
shows $A\ re\ \Delta = Some\ ((ctxt\text{-}of\text{-}pos\text{-}term\ q\ A)\langle(to\text{-}pterm\ (rhs\ \alpha)) \cdot \langle As\rangle_\alpha\rangle)$
⟨*proof*⟩

lemma *residual-well*:
$the\ (A\ re\ \Delta) \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *target-residual*:$target\ (the\ (A\ re\ \Delta)) = target\ A$
⟨*proof*⟩

lemma *deletion*:
shows $A -_p \Delta = Some\ ((ctxt\text{-}of\text{-}pos\text{-}term\ q\ A)\langle(to\text{-}pterm\ (lhs\ \alpha)) \cdot \langle As\rangle_\alpha\rangle)$
⟨*proof*⟩

lemma *deletion-well*:
shows $the\ (A -_p \Delta) \in wf\text{-}pterm\ R$
⟨*proof*⟩

end

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

interpretation *residual-op*:*op-proof-term R residual* ⟨*proof*⟩

lemma *a*:$A = (ctxt\text{-}of\text{-}pos\text{-}term\ q\ A)\langle(to\text{-}pterm\ (lhs\ \alpha)) \cdot \sigma\rangle$
⟨*proof*⟩

lemma *lin-lhs*:*linear-term* $(lhs\ \alpha)$
⟨*proof*⟩

lemma *is-fun-lhs*:*is-Fun* $(lhs\ \alpha)$
⟨*proof*⟩

abbreviation $As \equiv map\ \sigma\ (var\text{-}rule\ \alpha)$

lemma *lhs-subst*: $(to\text{-}pterm\ (lhs\ \alpha)) \cdot \sigma = (to\text{-}pterm\ (lhs\ \alpha)) \cdot \langle As\rangle_\alpha$
⟨*proof*⟩

lemma *rhs-subst*: $(to\text{-}pterm\ (rhs\ \alpha)) \cdot \sigma = (to\text{-}pterm\ (rhs\ \alpha)) \cdot \langle As\rangle_\alpha$
⟨*proof*⟩

lemma *as-well*:$\forall\, i < length\ As.\ As!i \in wf\text{-}pterm\ R$
⟨*proof*⟩

lemma *source-at-pq*:$source\ (A|\text{-}q) = (source\ A)|\text{-}p$
⟨*proof*⟩

lemma *single-redex-pterm*:
shows $\Delta = (ctxt\text{-}of\text{-}pos\text{-}term\ p\ (to\text{-}pterm\ (source\ A)))\langle Prule\ \alpha\ (map\ (to\text{-}pterm \circ source)\ As)\rangle$
⟨*proof*⟩

lemma *residual*:
shows A *re* Δ = *Some* ((*ctxt-of-pos-term q A*)⟨(*to-pterm* (*rhs* α)) · σ⟩)
⟨*proof*⟩

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* ⟹ ∀ *i* < *length As*. *As*!*i* $\perp_p$ *Bs*!*i* ⟹ *Fun f As* $\perp_p$ *Fun f Bs*
| *length As* = *length Bs* ⟹ ∀(*a*,*b*) ∈ *set*(*zip As Bs*). *a* $\perp_p$ *b* ⟹ (*Prule* α *As*) $\perp_p$ (*to-pterm* (*lhs* α)) · ⟨*Bs*⟩$_\alpha$
| *length As* = *length Bs* ⟹ ∀(*a*,*b*) ∈ *set*(*zip As Bs*). *a* $\perp_p$ *b* ⟹ (*to-pterm* (*lhs* α)) · ⟨*As*⟩$_\alpha$ $\perp_p$ (*Prule* α *Bs*)
lemmas *orthogonal.intros*[*intro*]

lemma *orth-symp*: *symp* ($\perp_p$)
⟨*proof*⟩

lemma *equal-imp-orthogonal*:
shows A $\perp_p$ A
⟨*proof*⟩

lemma *source-orthogonal*:
assumes *source* A = t
shows A $\perp_p$ *to-pterm* t
⟨*proof*⟩

lemma *orth-imp-co-initial*:
assumes A $\perp_p$ B
shows *co-initial* A B
⟨*proof*⟩

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 $\perp_p$ B
and A ∈ *wf-pterm R* **and** B ∈ *wf-pterm S*

shows A *re* $B \neq$ *None*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *orthogonal-ctxt*:
assumes $C\langle A\rangle \perp_p C\langle B\rangle$ $C \in$ *wf-pterm-ctxt* R
shows $A \perp_p B$
⟨*proof*⟩

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 $A \perp_p B$
⟨*proof*⟩

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* $\times$ *nat*$)$ *option*
type-synonym
$('f, 'v)$ *term-lab* $= ('f \times ('f, 'v)$ *label*$, 'v)$ *term*

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

abbreviation *labeled-lhs* $:: ('f, 'v)$ *prule* $\Rightarrow ('f, 'v)$ *term-lab*
where *labeled-lhs* $\alpha \equiv$ *label-term* α *0* (*lhs* α)

fun *labeled-source* $:: ('f, 'v)$ *pterm* $\Rightarrow ('f, 'v)$ *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) $\cdot \langle$*map labeled-source As*$\rangle_\alpha$

fun *term-lab-to-term* $:: ('f, 'v)$ *term-lab* $\Rightarrow ('f, 'v)$ *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* $\Rightarrow ('f, 'v)$ *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* $\Rightarrow ('f, 'v)$ *label*
where
get-label (*Var x*) $=$ *None*
| *get-label* (*Fun f ts*) $=$ *snd f*

fun *labelposs* $:: ('f, 'v)$ *term-lab* $\Rightarrow$ *pos set*
where
labelposs (*Var x*) $= \{\}$
| *labelposs* (*Fun* $(f,$ *None*$)$ *ts*$) = (\bigcup i<$*length ts*$. \{i \# p \mid p.\ p \in$ *labelposs* $(ts\ !\ i)\})$
| *labelposs* (*Fun* $(f,$ *Some l*$)$ *ts*$) = \{[]\} \cup (\bigcup i<$*length ts*$. \{i \# p \mid p.\ p \in$ *labelposs*

$(ts\ !\ i)\})$

abbreviation *possL* :: $('f, 'v)$ *pterm* $\Rightarrow$ *pos set*
where *possL* $A \equiv$ *labelposs* (*labeled-source* A)

lemma *labelposs-term-to-term-lab*: *labelposs* (*term-to-term-lab* t) $= \{\}$
⟨*proof*⟩

lemma *term-lab-to-term-lab*[*simp*]: *term-lab-to-term* (*term-to-term-lab* t) $= t$
⟨*proof*⟩

lemma *term-lab-to-term-subt-at*:
assumes $p \in$ *poss* t
shows *term-lab-to-term* $t\ |$-$p =$ *term-lab-to-term* $(t|$-$p)$
⟨*proof*⟩

lemma *vars-term-labeled-lhs*: *vars-term* (*label-term* $\alpha\ i\ t$) $=$ *vars-term* t
⟨*proof*⟩

lemma *vars-term-list-labeled-lhs*: *vars-term-list* (*label-term* $\alpha\ i\ t$) $=$ *vars-term-list* t
⟨*proof*⟩

lemma *var-poss-list-labeled-lhs*: *var-poss-list* (*label-term* $\alpha\ i\ t$) $=$ *var-poss-list* t
⟨*proof*⟩

lemma *var-labeled-lhs*[*simp*]: *vars-distinct* (*label-term* $\alpha\ i\ t$) $=$ *vars-distinct* t
⟨*proof*⟩

lemma *labelposs-subt-at*:
assumes $q \in$ *poss* t $p \in$ *labelposs* $(t|$-$q)$
shows $q@p \in$ *labelposs* t
⟨*proof*⟩

lemma *var-label-term*:
assumes $p \in$ *poss* t **and** $t|$-$p =$ *Var* x
shows *label-term* $\alpha\ n\ t\ |$-$p =$ *Var* x
⟨*proof*⟩

lemma *get-label-label-term*:
assumes $p \in$ *fun-poss* t
shows *get-label* (*label-term* $\alpha\ n\ t|$-$p)$ $=$ *Some* $(\alpha, n +$ *size* $p)$
⟨*proof*⟩

lemma *linear-label-term*:
assumes *linear-term* t
shows *linear-term* (*label-term* $\alpha\ n\ t$)
⟨*proof*⟩

lemma *var-term-lab-to-term*:
assumes $p \in$ *poss* t **and** $t|\text{-}p =$ *Var* x
shows *term-lab-to-term* t $|\text{-}p =$ *Var* x
⟨*proof*⟩

lemma *poss-term-lab-to-term*[*simp*]: *poss* $t =$ *poss* (*term-lab-to-term* t)
⟨*proof*⟩

lemma *fun-poss-term-lab-to-term*[*simp*]: *fun-poss* $t =$ *fun-poss* (*term-lab-to-term* t)
⟨*proof*⟩

lemma *vars-term-list-term-lab-to-term*: *vars-term-list* $t =$ *vars-term-list* (*term-lab-to-term* t)
⟨*proof*⟩

lemma *vars-term-list-term-to-term-lab*: *vars-term-list* (*term-to-term-lab* t) = *vars-term-list* t
⟨*proof*⟩

lemma *linear-term-to-term-lab*:
assumes *linear-term* t
shows *linear-term* (*term-to-term-lab* t)
⟨*proof*⟩

lemma *var-poss-list-term-lab-to-term*: *var-poss-list* $t =$ *var-poss-list* (*term-lab-to-term* t)
⟨*proof*⟩

lemma *label-poss-labeled-lhs*:
assumes $p \in$ *fun-poss* (*label-term* α n t)
shows $p \in$ *labelposs* (*label-term* α n t)
⟨*proof*⟩

lemma *labeled-var*:
assumes *source* $A =$ *Var* x
shows *labeled-source* $A =$ *Var* x
⟨*proof*⟩

lemma *labelposs-subs-fun-poss*: *labelposs* $t \subseteq$ *fun-poss* t
⟨*proof*⟩

lemma *labelposs-subs-poss*[*simp*]: *labelposs* $t \subseteq$ *poss* t
⟨*proof*⟩

lemma *get-label-imp-labelposs*:
assumes $p \in$ *poss* t **and** *get-label* $(t|\text{-}p) \neq$ *None*
shows $p \in$ *labelposs* t
⟨*proof*⟩

lemma *labelposs-obtain-label*:
assumes $p \in$ *labelposs* t
shows $\exists \alpha\ m.$ *get-label* $(t|\text{-}p) =$ *Some*(α, m)
⟨*proof*⟩

lemma *possL-obtain-label*:
assumes $p \in$ *possL* A
shows $\exists \alpha\ m.$ *get-label* $((\text{labeled-source } A)|\text{-}p) =$ *Some*(α, m)
⟨*proof*⟩

lemma *labeled-source-apply-subst*:
assumes $A \in$ *wf-pterm* R
shows *labeled-source* $(A \cdot \sigma) = (\text{labeled-source } A) \cdot (\text{labeled-source} \circ \sigma)$
⟨*proof*⟩

lemma *labelposs-apply-subst*:
labelposs $(s \cdot \sigma) =$ *labelposs* $s \cup \{p@q|\ p\ q\ x.\ p \in \text{var-poss } s \wedge s|\text{-}p = \text{Var } x \wedge q \in \text{labelposs } (\sigma\ x)\}$
⟨*proof*⟩

lemma *possL-apply-subst*:
assumes $A \cdot \sigma \in$ *wf-pterm* R
shows *possL* $(A \cdot \sigma) =$ *possL* $A \cup \{p@q|\ p\ q\ x.\ p \in \text{var-poss } (\text{labeled-source } A) \wedge (\text{labeled-source } A)|\text{-}p = \text{Var } x \wedge q \in \text{possL } (\sigma\ x)\}$
⟨*proof*⟩

lemma *label-term-to-term*[*simp*]: *term-lab-to-term* (*label-term* $\alpha\ n\ t) = t$
⟨*proof*⟩

lemma *fun-poss-label-term*: $p \in$ *fun-poss* (*label-term* $\beta\ n\ t) \longleftrightarrow p \in$ *fun-poss* t
⟨*proof*⟩

lemma *term-lab-to-term-subst*: *term-lab-to-term* $(t \cdot \sigma) =$ *term-lab-to-term* $t \cdot$ (*term-lab-to-term* $\circ\ \sigma$)
⟨*proof*⟩

lemma *labeled-source-to-term*[*simp*]: *term-lab-to-term* (*labeled-source* $A) =$ *source* A
⟨*proof*⟩

lemma *possL-subset-poss-source*: *possL* $A \subseteq$ *poss* (*source* A)
⟨*proof*⟩

lemma *labeled-source-pos*:
assumes $p \in$ *poss* s **and** *term-lab-to-term* $t = s$
shows *term-lab-to-term* $(t|\text{-}p) = s|\text{-}p$
⟨*proof*⟩

lemma *get-label-at-fun-poss-subst*:
assumes $p \in$ *fun-poss* t
shows *get-label* $((t \cdot \sigma)|\text{-}p) =$ *get-label* $(t|\text{-}p)$
⟨*proof*⟩

lemma *labeled-source-simple-pterm:possL* $($*to-pterm* $t) = \{\}$
⟨*proof*⟩

lemma *label-term-increase*:
assumes $s =$ (*label-term* α n $t) \cdot \sigma$ **and** $p \in$ *fun-poss* t
shows *get-label* $(s|\text{-}p) =$ *Some* $(\alpha,\ n +$ *length* $p)$
⟨*proof*⟩

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

lemma *label-term-max-value*:
assumes $p \in$ *poss* (*labeled-source* A) **and** *get-label* ((*labeled-source* $A)|\text{-}p) =$ *Some* $(\alpha,\ n)$
and $A \in$ *wf-pterm* R
shows $n \leq$ *length* p
⟨*proof*⟩

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

lemma *label-decrease*:
assumes $p@q \in$ *poss* (*labeled-source* A)
and *get-label* ((*labeled-source* $A)|\text{-}(p@q)) =$ *Some* $(\alpha,$ *length* $q + n)$
and $A \in$ *wf-pterm* R
shows *get-label* ((*labeled-source* $A)|\text{-}p) =$ *Some* $(\alpha,\ n)$
⟨*proof*⟩

If a function symbol is labeled with $(\alpha,\ n)$, then the function symbol n positions above it is labeled with $(\alpha,\ 0)$.

lemma *obtain-label-root*:
assumes $p \in$ *poss* (*labeled-source* A)
and *get-label* ((*labeled-source* $A)|\text{-}p) =$ *Some* $(\alpha,\ n)$
and $A \in$ *wf-pterm* R
shows *get-label* ((*labeled-source* $A)|\text{-}($*take* $($*length* $p - n)\ p)) =$ *Some* $(\alpha,\ 0) \wedge$ *take* $($*length* $p - n)\ p \in$ *poss* (*labeled-source* A)
⟨*proof*⟩

lemma *label-ctxt-apply-term*:
assumes *get-label* (*labeled-source* A |- $p) = l$ $q \in$ *poss* s
shows *get-label* (*labeled-source* ((*ctxt-of-pos-term* q (*to-pterm* $s)) \langle A\rangle)$ |- $(q@p))$ $= l$
⟨*proof*⟩

lemma *single-redex-at-p-label*:

assumes $p \in$ *poss* s **and** *is-Fun* (*lhs* α)
shows *get-label* (*labeled-source* (*ll-single-redex* s p α) |-p) = *Some*(α, *0*)
⟨*proof*⟩

Whenever a function symbol at position p has label (α, *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-source* A)
and *get-label* ((*labeled-source* A)|-p) = *Some* (α, *0*)
and $A \in$ *wf-pterm* R
shows $\exists\, q \in$ *poss* A. *ctxt-of-pos-term* p (*source* A) = *source-ctxt* (*ctxt-of-pos-term* q A) $\wedge$
A|-q = *Prule* α (*map* (λi. A|-(q@[i])) [*0*..<*length* (*var-rule* α)])
⟨*proof*⟩

lemma *poss-labeled-source-None*:
assumes $p \in$ *poss* (*labeled-source* A)
and *get-label* ((*labeled-source* A)|-p) = *None*
and $A \in$ *wf-pterm* R
shows $\exists\, q \in$ *poss* A. *ctxt-of-pos-term* p (*source* A) = *source-ctxt* (*ctxt-of-pos-term* q A)
⟨*proof*⟩
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*:$p \in$ *possL* (*to-pterm* $t \cdot \sigma$) **and** *well*:$\forall\, x \in$ *vars-term* t. $\sigma\ x \in$ *wf-pterm* R
shows $\exists\, p1\ p2\ x\ \gamma$. $p1 \in$ *poss* $t \wedge t$|-$p1$ = *Var* $x \wedge p1$@$p2 \leq_p p$
$\wedge\ p2 \in$ *possL* ($\sigma\ x$) $\wedge$ *get-label* ((*labeled-source* ($\sigma\ x$))|-$p2$) = *Some* (γ, *0*)
⟨*proof*⟩

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* ($\sigma\ x$))
⟨*proof*⟩

lemma *set-labelposs-subst*:
labelposs ($t \cdot \sigma$) = *labelposs* $t \cup$ ($\bigcup i <$ *length* (*vars-term-list* t). {(*var-poss-list* t!i)@q | q. $q \in$ *labelposs* (σ (*vars-term-list* t ! i))}) (**is** *?ps* = *?qs*)

⟨*proof*⟩

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) ∪ ($\bigcup i<$ (*length As*). {(*var-poss-list* (*lhs* α)!*i*)@*q* | *q*. *q* ∈ *possL*(*As*!*i*)})
⟨*proof*⟩

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

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* ∈ *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 *labeled-source* (*ctxt-of-pos-term p*′ *A*)⟨*B*⟩ = (*ctxt-of-pos-term p* (*labeled-source A*))⟨*labeled-source B*⟩
⟨*proof*⟩
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*)
⟨*proof*⟩

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*}
⟨*proof*⟩

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* $\alpha)\}$
⟨*proof*⟩

end

lemma *labeled-poss-in-lhs*:
assumes *p-pos*:$p \in$ *poss* (*source* (*Prule* α *As*)) **and** *well*:*Prule* α *As* $\in$ *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*⟩

context *left-lin-no-var-lhs*
begin
lemma *get-label-Prule*:
assumes *Prule* α *As* $\in$ *wf-pterm* R **and** $p \in$ *poss* (*source* (*Prule* α *As*)) **and** *get-label* (*labeled-source* (*Prule* α *As*) |- p) = *Some* (β, 0)
shows ($p = [] \wedge \alpha = \beta$) $\vee$
($\exists$ *p1 p2 i.* p = *p1*@*p2* $\wedge$ $i <$ *length As* $\wedge$ *var-poss-list* (*lhs* α)!i = *p1* $\wedge$
p2 $\in$ *poss* (*source* (*As*!i)) $\wedge$ *get-label* (*labeled-source* (*As*!i)|-*p2*) = *Some* (β, 0))
⟨*proof*⟩
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))
⟨*proof*⟩

lemma *unlabeled-source-to-pterm*:
assumes *labeled-source* $A = s \cdot \tau$
and *linear-term* s **and** $A \in$ *wf-pterm* R
and *labelposs* $s = \{\}$
shows $\exists$ *As.* A = *to-pterm* (*term-lab-to-term* s) $\cdot$ (*mk-subst Var* (*zip* (*vars-term-list* s) *As*)) $\wedge$ *length* (*vars-term-list* s) = *length As*
⟨*proof*⟩
end

lemma *labels-intersect-label-term*:
assumes *term-lab-to-term* $A = t \cdot$ (*term-lab-to-term* $\circ\ \sigma$)

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩
end

lemma (**in** *single-redex*) *labeled-source-at-pq*:*labeled-source* (*A*|-*q*) = (*labeled-source A*)|-*p*
⟨*proof*⟩

context *left-lin*
begin
lemma *single-redex-label*:
assumes Δ = *ll-single-redex s p* α *p* ∈ *poss s q* ∈ *poss* (*source* Δ) *to-rule* α ∈ *R*
and *get-label* (*labeled-source* Δ |-*q*) = *Some* (β, *n*)
shows α = β ∧ (∃ *q*′. *q* = *p*@*q*′ ∧ *length q*′ = *n* ∧ *q*′ ∈ *fun-poss* (*lhs* α))
⟨*proof*⟩
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*)
⟨*proof*⟩

lemma *finite-possL*: *finite* (*possL A*)
⟨*proof*⟩

lemma *measure-ov-symm*: *measure-ov A B* = *measure-ov B A*
⟨*proof*⟩

lemma *measure-lhs-subst*:
assumes *l*:*length As* = *length Bs*
shows *card* ((*labelposs* ((*label-term* α *j t*) · $\langle$*map labeled-source As*$\rangle_\alpha$)) ∩
(*labelposs* (*labeled-source* (*to-pterm t*) · $\langle$*map labeled-source Bs*$\rangle_\alpha$)))
= ($\sum$ *x*←*vars-term-list t*. *measure-ov* (($\langle$*As*$\rangle_\alpha$) *x*) (($\langle$*Bs*$\rangle_\alpha$) *x*))
⟨*proof*⟩

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)) · $\langle$*Bs*$\rangle_\alpha$) = *0*
⟨*proof*⟩

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*) = {}
⟨*proof*⟩

lemma *empty-step-imp-measure-zero*:
assumes *is-empty-step A*
shows *measure-ov A B* = *0*
⟨*proof*⟩

lemma *measure-ov-to-pterm*:
shows *measure-ov A* (*to-pterm t*) = *0*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

Define the partial order on overlaps

definition *less-eq-overlap* :: *pos* × *pos* ⇒ *pos* × *pos* ⇒ *bool* (**infix** ≤$_o$ *50*)
where *p* ≤$_o$ *q* ⟷ (*fst p* ≤$_p$ *fst q*) ∧ (*snd p* ≤$_p$ *snd q*)

definition *less-overlap* :: *pos* × *pos* ⇒ *pos* × *pos* ⇒ *bool* (**infix** <$_o$ *50*)
where *p* <$_o$ *q* ⟷ *p* ≤$_o$ *q* ∧ *p* ≠ *q*

interpretation *order-overlaps*: *order less-eq-overlap less-overlap*
⟨*proof*⟩

lemma *overlaps-pos-finite*: *finite* (*overlaps-pos A B*)
⟨*proof*⟩

lemma *labeled-sources-imp-measure-not-zero*:
assumes *p* ∈ *poss* (*labeled-source A*) *p* ∈ *poss* (*labeled-source B*)
and *get-label* ((*labeled-source A*)|-*p*) ≠ *None* ∧ *get-label* ((*labeled-source B*)|-*p*)
≠ *None*
shows *measure-ov A B* > *0*
⟨*proof*⟩

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*) = {}
⟨*proof*⟩

lemma *empty-overlaps-imp-measure-zero*:
assumes *A* ∈ *wf-pterm R* **and** *B* ∈ *wf-pterm S*
and *overlaps-pos* (*labeled-source A*) (*labeled-source B*) = {}
shows *measure-ov A B* = *0*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *lexord-prefix-diff*:
assumes (*ord.lexordp* ((<) :: *nat* ⇒ *nat* ⇒ *bool*)) *xs ys* **and** ¬ *prefix xs ys*
shows (*ord.lexordp* (<)) (*xs*@*us*) (*ys*@*vs*)
⟨*proof*⟩

lemma *var-poss-list-sorted*: *sorted-wrt* (*ord.lexordp* ((<) :: *nat* ⇒ *nat* ⇒ *bool*)) (*var-poss-list t*)
⟨*proof*⟩

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

lemma *redex-patterns-sorted*:
assumes $A \in$ *wf-pterm* R
shows *sorted-wrt* (*ord.lexordp* (<)) (*map snd* (*redex-patterns* A))
⟨*proof*⟩

corollary *distinct-snd-rdp*:
assumes $A \in$ *wf-pterm* R
shows *distinct* (*map snd* (*redex-patterns* A))
⟨*proof*⟩

lemma *redex-patterns-equal*:
assumes *wf*:$A \in$ *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*⟩

lemma *redex-patterns-order*:
assumes $A \in$ *wf-pterm* R **and** $i < j$ **and** $j <$ *length* (*redex-patterns* A)
and *redex-patterns* A ! $i = (\alpha, p1)$ **and** *redex-patterns* A ! $j = (\beta, p2)$
shows ¬ $p2 \leq_p p1$
⟨*proof*⟩

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* $(\alpha, 0)$
⟨*proof*⟩

lemma *redex-pattern-rule-symbol*:
assumes $A \in$ *wf-pterm* R $(\alpha, p) \in$ *set* (*redex-patterns* A)
shows *to-rule* $\alpha \in R$
⟨*proof*⟩

lemma *redex-patterns-subset-possL*:
assumes $A \in$ *wf-pterm* R
shows *set* (*map snd* (*redex-patterns* A)) $\subseteq$ *possL* A
⟨*proof*⟩
end

lemma *redex-poss-empty-imp-empty-step*:
assumes *redex-patterns* A = []
shows *is-empty-step* A
⟨*proof*⟩

lemma *overlap-imp-redex-poss*:
assumes $A \in$ *wf-pterm* R $B \in$ *wf-pterm* R
and *measure-ov* A $B \neq 0$
shows *redex-patterns* $A \neq []$
⟨*proof*⟩

lemma *redex-patterns-to-pterm*:
shows *redex-patterns* (*to-pterm* s) $= []$
⟨*proof*⟩

lemma *redex-patterns-elem-fun*:
assumes $(\alpha, p) \in$ *set* (*redex-patterns* (*Pfun* f *As*))
shows $\exists i\ p'.\ i <$ *length As* $\wedge\ p = i\#p' \wedge (\alpha, p') \in$ *set* (*redex-patterns* (*As*!i))
⟨*proof*⟩

lemma *redex-patterns-elem-rule*:
assumes $(\alpha, p) \in$ *set* (*redex-patterns* (*Prule* β *As*))
shows $p = [] \vee (\exists i\ p'.\ i <$ *length As* $\wedge\ i <$ *length* (*var-poss-list* (*lhs* β))
$\wedge\ p =$ (*var-poss-list* (*lhs* β)!i)@$p' \wedge (\alpha, p') \in$ *set* (*redex-patterns* (*As*!i)))
⟨*proof*⟩

lemma *redex-patterns-elem-fun'*:
assumes $(\alpha, p) \in$ *set* (*redex-patterns* (*As*!i)) **and** i:$i <$ *length As*
shows $(\alpha, i\#p) \in$ *set* (*redex-patterns* (*Pfun* f *As*))
⟨*proof*⟩

lemma *redex-patterns-elem-rule'*:
assumes $(\beta, p) \in$ *set* (*redex-patterns* (*As*!i)) **and** i:$i <$ *length As* $i <$ *length* (*var-poss-list* (*lhs* α))
shows $(\beta,$ (*var-poss-list* (*lhs* α) ! i)@$p) \in$ *set* (*redex-patterns* (*Prule* α *As*))
⟨*proof*⟩

lemma *redex-patterns-elem-subst*:
assumes $(\alpha, p) \in$ *set* (*redex-patterns* ((*to-pterm* t) $\cdot\ \sigma$))
shows $\exists$ *p1 p2 x.* $p =$ *p1*@*p2* $\wedge$ $(\alpha,$ *p2*$) \in$ *set* (*redex-patterns* (σ x)) $\wedge$ *p1* $\in$ *var-poss t* $\wedge$ $t|$-*p1* $=$ *Var x*
⟨*proof*⟩

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*⟩

lemma *redex-patterns-elem-subst'*:
assumes (α, *p2*) ∈ *set* (*redex-patterns* (σ *x*)) **and** *p1*:*p1* ∈ *poss t t*|-*p1* = *Var x*
shows (α, *p1*@*p2*) ∈ *set* (*redex-patterns* ((*to-pterm t*) · σ))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *redex-pattern-proof-term-equality*:
assumes *A* ∈ *wf-pterm R B* ∈ *wf-pterm R*
and *set* (*redex-patterns A*) = *set* (*redex-patterns B*)
and *source A* = *source B*
shows *A* = *B*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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

lemma *source-single-step*:
assumes Δ:Δ ∈ *set* (*single-steps A*) **and** *wf*:*A* ∈ *wf-pterm R*
shows *source* Δ = *source A*
⟨*proof*⟩

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]
⟨*proof*⟩

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*⟩

lemma *single-steps-measure*:
assumes Δ*1* :Δ*1* ∈ *set* (*single-steps A*) **and** Δ*2*:Δ*2* ∈ *set* (*single-steps A*)
and *wf*:*A* ∈ *wf-pterm R* **and** *neq*:Δ*1* ≠ Δ*2*
shows *measure-ov* Δ*1* Δ*2* = *0*
⟨*proof*⟩

lemma *single-steps-orth*:
assumes Δ*1*:Δ*1* ∈ *set* (*single-steps A*) **and** Δ*2*:Δ*2* ∈ *set* (*single-steps A*) **and**

wf:*A* ∈ *wf-pterm R*
shows Δ*1* ⊥$_p$ Δ*2*
⟨*proof*⟩

lemma *redex-patterns-below*:
assumes *wf*:*A* ∈ *wf-pterm R*
and (α, *p*) ∈ *set* (*redex-patterns A*)
and (β, *p*@*q*) ∈ *set* (*redex-patterns A*) **and** *q* ≠ []
shows *q* ∉ *fun-poss* (*lhs* α)
⟨*proof*⟩

lemma *single-steps-singleton*:
assumes *A-wf*:*A* ∈ *wf-pterm R* **and** Δ:*single-steps A* = [Δ]
shows *A* = Δ
⟨*proof*⟩
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*⟩
end

context *left-lin-no-var-lhs*
begin
lemma *label-single-step*:
assumes *p* ∈ *poss* (*labeled-source A*) *A* ∈ *wf-pterm R*
and *get-label* (*labeled-source A* |- *p*) = *Some* (α, *n*)
shows ∃ *Ai*. *Ai* ∈ *set* (*single-steps A*) ∧ *get-label* (*labeled-source Ai* |- *p*) = *Some* (α, *n*)
⟨*proof*⟩

lemma *proof-term-matches*:
assumes *A* ∈ *wf-pterm R B* ∈ *wf-pterm R linear-term A*
and ⋀α *r*. (α, *r*) ∈ *set* (*redex-patterns A*) = ((α, *r*) ∈ *set* (*redex-patterns B*) ∧ *r* ∈ *fun-poss* (*source A*))
and *source A* · σ = *source B*
shows *A* · (*mk-subst Var* (*match-substs A B*)) = *B*
⟨*proof*⟩
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*⟩

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*⟩
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 International 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.