

Nominal Unification

Guilherme Borges Brandão¹ Thomas Ammer²
Daniele Nantes Sobrinho¹ Mauricio Ayala-Rinción¹
Christian Urban² Maribel Fernández²
Mohammad Abdulaziz²

¹ Universidade de Brasília, Brasília D.F., Brazil

² King's College London, London, U.K.

May 8, 2026

Abstract

This is an improved and updated formalisation of the nominal unification algorithm. Nominal unification is a generalisation of the classic first-order unification to the practically important case of equations between terms involving binding operations. A simple, possibly capturing substitution of terms for variables solves such equations if it makes the equated terms alpha-equivalent, i.e. equal up to renaming bound names. While nominal unification can be seen as equivalent to Miller's higher-order pattern unification (unification problems can be inter-coded), nominal unification has the advantage of involving only first-order syntax. This means in the presented formalisation we only need to reason about standard inductive datatypes and first-order operations. The main improvement of this development is a much simpler proof for establishing the equivalence relation properties for an inductive definition characterising when two terms are alpha-equivalent. In the original literature, this involved a clunky mutual induction over the size of terms involving three properties. Here we can separate the proofs and use just structural inductions. This results in a much smoother formalisation of the nominal unification algorithm.

Contents

1	Introduction	2
2	Swappings of Pairs of Atoms	2
3	Terms	3
4	Facts about Atoms Occurring in Swappings	6

5	Disagreement Sets	7
6	Freshness	9
7	Pre-Equality	10
8	Equality	12
9	Substitutions	13
10	Most General Unifiers	16
11	Termination	21
12	Unification	23
	References	25

1 Introduction

This is an improved and updated formalisation of the nominal unification algorithm given in [5, 6]. The formalisation presented here is improved according to the Rocha Oliveira et al. approach formalised in PVS [2, 4], where the properties of symmetry, transitivity, and equivariance of the α -equivalence relation are proved separately. This makes the proof and formalisation more convenient. The same approach was taken by de Carvalho et al. in the Coq formalisation of nominal α -unification and nominal C-unification [1, 3]. In the PVS formalisation, nominal unification is presented as a functional recursive algorithm, proved sound and complete, from which executable code in Lisp can be extracted. In the Coq formalisation, non-deterministic rule-based nominal unification procedures were presented, following the paper by Urban et al, which were proved sound and complete; further, recursive definitions were presented, which were proved equivalent to the inductive procedures, and from which executable code was extracted.

2 Swappings of Pairs of Atoms

```
fun swapa :: (string × string) ⇒ string ⇒ string
where
  swapa (b,c) a = (if b = a then c else (if c = a then b else a))
```

```
fun swapas:: (string × string) list ⇒ string ⇒ string
where
  swapas [] a = a
| swapas (x#pi) a = swapa x (swapas pi a)
```

lemma *swapas-aa*[*simp*]:
shows *swapas* [(*a,a*)] *b* = *b*
 ⟨*proof*⟩

lemma *swapas-ab-ba*:
shows *swapas* [(*a,b*)] = *swapas* [(*b,a*)]
 ⟨*proof*⟩

lemma *swapas-append*:
shows *swapas* (*pi1*@*pi2*) *a* = *swapas pi1* (*swapas pi2 a*)
 ⟨*proof*⟩

lemma *swapas-inv* [*simp*]:
shows *swapas* (*rev pi*) (*swapas pi a*) = *a*
 ⟨*proof*⟩

lemma *swapas-rev-pi-a*:
assumes *swapas pi a* = *b*
shows *swapas* (*rev pi*) *b* = *a*
 ⟨*proof*⟩

lemma *swapas-rev-swapas-id* [*simp*]:
shows *swapas pi* (*swapas* (*rev pi*) *a*) = *a*
 ⟨*proof*⟩

lemma *swapas-rev-pi-b*:
assumes *swapas* (*rev pi*) *a* = *b*
shows *swapas pi b* = *a*
 ⟨*proof*⟩

lemma *swapas-comm*:
shows (*swapas* (*pi*@[(*a,b*))] *c*) = (*swapas* [(*swapas pi a,swapas pi b*)@*pi*] *c*)
 ⟨*proof*⟩

3 Terms

Definitions for terms, occurs check and the notion of subterms.

datatype *trm* = *Abst string trm*
 | *Susp (string × string) list string*
 | *Unit*
 | *Atom string*
 | *Paar trm trm*
 | *Func string trm*

The swapping operation on terms.

fun *swap* :: (*string × string*) *list* ⇒ *trm* ⇒ *trm*
where

```

  swap pi (Unit)      = Unit
| swap pi (Atom a)   = Atom (swapas pi a)
| swap pi (Susp pi' X) = Susp (pi @ pi') X
| swap pi (Abst a t) = Abst (swapas pi a) (swap pi t)
| swap pi (Paar t1 t2) = Paar (swap pi t1) (swap pi t2)
| swap pi (Func F t) = Func F (swap pi t)

```

lemma *swap-id* [simp]:
shows $swap \ [] \ t = t$
 ⟨proof⟩

lemma *swap-append*:
shows $swap \ (pi1@pi2) \ t = swap \ pi1 \ (swap \ pi2 \ t)$
 ⟨proof⟩

lemma *swap-empty*:
assumes $swap \ pi \ t = Susp \ [] \ X$
shows $pi = []$
 ⟨proof⟩

The depth of terms used as measure.

```

fun depth :: trm ⇒ nat
  where
    depth (Unit)      = 0
| depth (Atom a)     = 0
| depth (Susp pi X)  = 0
| depth (Abst a t)   = 1 + depth t
| depth (Func F t)   = 1 + depth t
| depth (Paar t t') = 1 + (max (depth t) (depth t'))

```

lemma
Suc-max-left: $Suc(max \ x \ y) = z \longrightarrow x < z$ **and**
Suc-max-right: $Suc(max \ x \ y) = z \longrightarrow y < z$
 ⟨proof⟩

lemma *swap-depth* [simp]:
shows $depth \ (swap \ pi \ t) = depth \ t$
 ⟨proof⟩

The occurs predicate and variables inside terms.

```

fun occurs :: string ⇒ trm ⇒ bool
  where
    occurs X (Unit)      = False
| occurs X (Atom a)     = False
| occurs X (Susp pi Y)  = (if X = Y then True else False)
| occurs X (Abst a t)   = occurs X t
| occurs X (Paar t1 t2) = (if (occurs X t1) then True else (occurs X t2))
| occurs X (Func F t)   = occurs X t

```

```

fun vars-trm :: trm  $\Rightarrow$  string set
  where
    vars-trm (Unit)      = {}
  | vars-trm (Atom a)    = {}
  | vars-trm (Susp pi X) = {X}
  | vars-trm (Paar t1 t2) = (vars-trm t1)  $\cup$  (vars-trm t2)
  | vars-trm (Abst a t)  = vars-trm t
  | vars-trm (Func F t)  = vars-trm t

```

lemma vars-swap:
shows vars-trm (swap pi t) = vars-trm t
 <proof>

Subterms and proper subterms.

```

fun sub-trms :: trm  $\Rightarrow$  trm set
  where
    sub-trms (Unit)      = {Unit}
  | sub-trms (Atom a)    = {Atom a}
  | sub-trms (Susp pi Y) = {Susp pi Y}
  | sub-trms (Abst a t)  = {Abst a t}  $\cup$  sub-trms t
  | sub-trms (Paar t1 t2) = {Paar t1 t2}  $\cup$  sub-trms t1  $\cup$  sub-trms t2
  | sub-trms (Func F t)  = {Func F t}  $\cup$  sub-trms t

```

```

fun psub-trms :: trm  $\Rightarrow$  trm set
  where
    psub-trms (Unit)      = {}
  | psub-trms (Atom a)    = {}
  | psub-trms (Susp pi X) = {}
  | psub-trms (Paar t1 t2) = sub-trms t1  $\cup$  sub-trms t2
  | psub-trms (Abst a t)  = sub-trms t
  | psub-trms (Func F t)  = sub-trms t

```

lemma psub-sub-trms:
assumes t1 \in psub-trms t2
shows t1 \in sub-trms t2
 <proof>

lemma t-sub-trms-t:
shows t \in sub-trms t
 <proof>

lemma abst-psub-trms:
assumes Abst a t1 \in sub-trms t2
shows t1 \in psub-trms t2
 <proof>

lemma func-psub-trms:
assumes Func F t1 \in sub-trms t2
shows t1 \in psub-trms t2

<proof>

lemma *paar-psub-trms*:

assumes *Paar t1 t2 ∈ sub-trms t3*

shows *t1 ∈ psub-trms t3 and t2 ∈ psub-trms t3*

<proof>

lemma *t-not-in-psub-trms-t*:

shows *t ∉ psub-trms t*

<proof>

lemma *if-sub-not-eq-then-psub*:

assumes *t1 ∈ sub-trms t2 t1 ≠ t2*

shows *t1 ∈ psub-trms t2*

<proof>

lemma *depth-psub-trms*:

assumes *t1 ∈ psub-trms t2*

shows *depth t1 < depth t2*

<proof>

4 Facts about Atoms Occurring in Swappings

fun *atms* :: *(string × string) list ⇒ string set where*

atms [] = {} |

atms (x#xs) = ((atms xs) ∪ {fst(x),snd(x)})

lemma *atms-append[simp]*:

shows *atms (xs@ys) = atms xs ∪ atms ys*

<proof>

lemma *atms-rev[simp]*:

shows *atms (rev pi) = atms pi*

<proof>

lemma *a-not-in-atms*:

assumes *a ∉ atms pi*

shows *a = swapas pi a*

<proof>

lemma *swapas-pi-ineq-a*:

assumes *swapas pi a ≠ a*

shows *a ∈ atms pi*

<proof>

lemma *a-ineq-swapas-pi*:

assumes *a ≠ swapas pi a*

shows *a ∈ atms pi*

<proof>

lemma *swapas-pi-in-atms*:
assumes $a \in \text{atms } pi$
shows $\text{swapas } pi \ a \in \text{atms } pi$
 $\langle \text{proof} \rangle$

5 Disagreement Sets

definition $ds :: (\text{string} \times \text{string}) \text{ list} \Rightarrow (\text{string} \times \text{string}) \text{ list} \Rightarrow \text{string set}$ **where**
 $ds\text{-def: } ds \ xs \ ys \equiv \{ a . a \in (\text{atms } xs \cup \text{atms } ys) \wedge (\text{swapas } xs \ a \neq \text{swapas } ys \ a) \}$

lemma *ds-elem*:
assumes $\text{swapas } pi \ a \neq a$
shows $a \in ds \ [] \ pi$
 $\langle \text{proof} \rangle$

corollary *ds-elem-cp*:
assumes $a \notin ds \ [] \ pi$
shows $\text{swapas } pi \ a = a$
 $\langle \text{proof} \rangle$

lemma *elem-ds*:
assumes $a \in ds \ [] \ pi$
shows $a \neq \text{swapas } pi \ a$
 $\langle \text{proof} \rangle$

lemma *ds-sym*:
shows $ds \ pi1 \ pi2 = ds \ pi2 \ pi1$
 $\langle \text{proof} \rangle$

lemma *ds-trans*:
assumes $c \in ds \ pi1 \ pi3$
shows $c \in ds \ pi1 \ pi2 \vee c \in ds \ pi2 \ pi3$
 $\langle \text{proof} \rangle$

lemma *ds-cancel-pi-left*:
assumes $c \in ds \ (pi1 @ pi) \ (pi2 @ pi)$
shows $\text{swapas } pi \ c \in ds \ pi1 \ pi2$
 $\langle \text{proof} \rangle$

lemma *ds-cancel-pi-right*:
assumes $\text{swapas } pi \ c \in ds \ pi1 \ pi2$
shows $c \in ds \ (pi1 @ pi) \ (pi2 @ pi)$
 $\langle \text{proof} \rangle$

lemma *ds-equality*:
shows $(ds \ [] \ pi) - \{a, \text{swapas } pi \ a\} = (ds \ [] \ ((a, \text{swapas } pi \ a) \# pi)) - \{\text{swapas } pi \ a\}$
 $\langle \text{proof} \rangle$

lemma *ds-7*:
assumes $b \neq \text{swapas } pi \ b \ a \in ds \ [] \ ((b, \text{swapas } pi \ b) \# pi)$
shows $a \in ds \ [] \ pi$
 $\langle proof \rangle$

lemma *ds-cancel-pi-front*:
shows $ds \ (pi @ pi1) \ (pi @ pi2) = ds \ pi1 \ pi2$
 $\langle proof \rangle$

lemma *ds-rev-pi-pi*:
shows $ds \ ((rev \ pi1) @ pi1) \ pi2 = ds \ [] \ pi2$
 $\langle proof \rangle$

lemma *ds-rev*:
shows $ds \ [] \ ((rev \ pi1) @ pi2) = ds \ pi1 \ pi2$
 $\langle proof \rangle$

lemma *ds-acabb*:
assumes $a \neq b \ b \neq c \ c \neq a$
shows $ds \ [(a, b), (b, c)] \ [(a, c)] = \{a, b\}$
 $\langle proof \rangle$

lemma *ds-baab*:
assumes $a \neq b$
shows $ds \ [(b, a)] \ [(a, b)] = \{\}$
 $\langle proof \rangle$

lemma *ds-baab-id*:
assumes $a \neq b$
shows $ds \ ((b, a) @ [(a, b)]) \ [] = \{\}$
 $\langle proof \rangle$

lemma *ds-abab*:
shows $ds \ [] \ [(a, b), (a, b)] = \{\}$
 $\langle proof \rangle$

lemma *ds-comm*:
shows $ds \ (pi @ [(a, b)]) \ ((\text{swapas } pi \ a, \text{swapas } pi \ b) @ pi) = \{\}$
 $\langle proof \rangle$

lemma *ds-rev-pi-id*:
shows $ds \ (rev \ pi @ pi) \ [] = \{\}$
 $\langle proof \rangle$

lemma *ds-pi-rev-id*:
shows $ds \ (pi @ rev \ pi) \ [] = \{\}$
 $\langle proof \rangle$

lemma *ds-swapas-eq*:
assumes $ds\ pi1\ pi2 = \{\}$
shows $swapas\ pi1\ a = swapas\ pi2\ a$
 $\langle proof \rangle$

Disagreement sets as lists.

fun *flatten* :: $(string \times string)list \Rightarrow string\ list$ **where**
 $flatten\ [] = []$ |
 $flatten\ (x\#\ xs) = (fst\ x)\#\ (snd\ x)\#\ (flatten\ xs)$

definition *ds-list* :: $(string \times string)list \Rightarrow (string \times string)list \Rightarrow string\ list$
where
ds-list-def: $ds\ list\ pi1\ pi2 \equiv remdups\ ([x.\ x <- (flatten\ (pi1\@\ pi2)),\ x \in ds\ pi1\ pi2])$

lemma *set-flatten-eq-atms*:
shows $set\ (flatten\ pi) = atms\ pi$
 $\langle proof \rangle$

lemma *ds-list-equ-ds*:
 $set\ (ds\ list\ pi1\ pi2) = ds\ pi1\ pi2$
 $\langle proof \rangle$

6 Freshness

Defines the freshness relation and shows facts about its behaviour under swappings.

type-synonym *fresh-envs* = $(string \times string)\ set$

inductive *fresh* :: $fresh\ envs \Rightarrow string \Rightarrow trm \Rightarrow bool$ (- \vdash - $\#$ - [80,80,80] 80)
where
fresh-abst-ab[*intro!*]: $\llbracket nabl\ a \vdash t; a \neq b \rrbracket \Longrightarrow nabl\ a \vdash Abst\ b\ t$ |
fresh-abst-aa[*intro!*]: $nabl\ a \vdash Abst\ a\ t$ |
fresh-unit[*intro!*]: $nabl\ a \vdash Unit$ |
fresh-atom[*intro!*]: $a \neq b \Longrightarrow nabl\ a \vdash Atom\ b$ |
fresh-susp[*intro!*]: $(swapas\ (rev\ pi)\ a, X) \in nabl\ a \Longrightarrow nabl\ a \vdash Susp\ pi\ X$ |
fresh-paar[*intro!*]: $\llbracket nabl\ a \vdash t1; nabl\ a \vdash t2 \rrbracket \Longrightarrow nabl\ a \vdash Paar\ t1\ t2$ |
fresh-func[*intro!*]: $nabl\ a \vdash t \Longrightarrow nabl\ a \vdash Func\ F\ t$

inductive-cases *Fresh-elim*s:

$nabl\ a \vdash Abst\ b\ t$
 $nabl\ a \vdash Unit$
 $nabl\ a \vdash Atom\ b$
 $nabl\ a \vdash Susp\ pi\ X$
 $nabl\ a \vdash Paar\ t1\ t2$
 $nabl\ a \vdash Func\ F\ t$

lemma *fresh-swap-eqv*t:

assumes $nabla \vdash a \# t$
shows $nabla \vdash \text{swapas } \pi a \# \text{swap } \pi t$
 $\langle \text{proof} \rangle$

lemma *fresh-swap-left*:
assumes $nabla \vdash a \# \text{swap } \pi t$
shows $nabla \vdash \text{swapas } (\text{rev } \pi) a \# t$
 $\langle \text{proof} \rangle$

lemma *fresh-swap-right*:
assumes $nabla \vdash \text{swapas } (\text{rev } \pi) a \# t$
shows $nabla \vdash a \# \text{swap } \pi t$
 $\langle \text{proof} \rangle$

lemma *fresh-weak*:
assumes $nabla1 \vdash a \# t$
shows $(nabla1 \cup nabla2) \vdash a \# t$
 $\langle \text{proof} \rangle$

lemma *ds-empty-fresh-1*:
assumes $ds \ \pi1 \ \pi2 = \{\}$
shows $nabla \vdash \text{swapas } \pi1 a \# t \implies nabla \vdash \text{swapas } \pi2 a \# t$
 $\langle \text{proof} \rangle$

lemma *ds-empty-fresh-2*:
assumes $ds \ \pi1 \ \pi2 = \{\}$
shows $nabla \vdash a \# \text{swap } \pi1 t \implies nabla \vdash a \# \text{swap } \pi2 t$
 $\langle \text{proof} \rangle$

7 Pre-Equality

Defines the relation which captures the notion of alpha-equivalence (on open terms) and proves this relation is an equivalence relation.

inductive *equ* :: $\text{fresh-envs} \Rightarrow \text{trm} \Rightarrow \text{trm} \Rightarrow \text{bool}$ ($- \vdash - \approx - [80, 80, 80] 80$) **where**
equ-abst-ab[*intro!*]: $\llbracket a \neq b; (nabla \vdash a \# t2); (nabla \vdash t1 \approx (\text{swap } [(a, b)] t2)) \rrbracket$
 $\implies (nabla \vdash \text{Abst } a t1 \approx \text{Abst } b t2) \mid$
equ-abst-aa[*intro!*]: $(nabla \vdash t1 \approx t2) \implies (nabla \vdash \text{Abst } a t1 \approx \text{Abst } a t2) \mid$
equ-unit[*intro!*]: $(nabla \vdash \text{Unit} \approx \text{Unit}) \mid$
equ-atom[*intro!*]: $a = b \implies nabla \vdash \text{Atom } a \approx \text{Atom } b \mid$
equ-susp[*intro!*]: $(\forall c \in ds \ \pi1 \ \pi2. (c, X) \in nabla) \implies (nabla \vdash \text{Susp } \pi1 X \approx \text{Susp } \pi2 X) \mid$
equ-paar[*intro!*]: $\llbracket (nabla \vdash t1 \approx t2); (nabla \vdash s1 \approx s2) \rrbracket \implies (nabla \vdash \text{Paar } t1 s1 \approx \text{Paar } t2 s2) \mid$
equ-func[*intro!*]: $(nabla \vdash t1 \approx t2) \implies (nabla \vdash \text{Func } F t1 \approx \text{Func } F t2)$

inductive-cases *Equ-elim*:
 $nabla \vdash \text{Atom } a \approx \text{Atom } b$

$nabla \vdash Unit \approx Unit$
 $nabla \vdash Susp\ pi1\ X \approx Susp\ pi2\ X$
 $nabla \vdash Paar\ s1\ t1 \approx Paar\ s2\ t2$
 $nabla \vdash Func\ F\ t1 \approx Func\ F\ t2$
 $nabla \vdash Abst\ a\ t1 \approx Abst\ a\ t2$
 $nabla \vdash Abst\ a\ t1 \approx Abst\ b\ t2$

lemma *equ-depth*:

assumes $nabla \vdash t1 \approx t2$
shows $depth\ t1 = depth\ t2$
 $\langle proof \rangle$

lemma *rev-pi-pi-equ*:

shows $(nabla \vdash swap\ (rev\ pi)\ (swap\ pi\ t) \approx t)$
 $\langle proof \rangle$

lemma *equ-pi-right*:

assumes $\forall a \in ds \ []\ pi.\ nabla \vdash a \# t$
shows $nabla \vdash t \approx swap\ pi\ t$
 $\langle proof \rangle$

lemma *pi-comm*:

shows $nabla \vdash (swap\ (pi\ @\ [(a,b)])\ t) \approx (swap\ [(swapas\ pi\ a,\ swapas\ pi\ b)]\ @\ pi)\ t)$
 $\langle proof \rangle$

lemma *l3-jud*:

assumes $(nabla \vdash t1 \approx t2)$
shows $(nabla \vdash a \# t1) \longrightarrow (nabla \vdash a \# t2)$
 $\langle proof \rangle$

lemma *ds-empty-equiv-1*:

assumes $ds\ pi1\ pi2 = \{\}$
shows $nabla \vdash swap\ pi1\ t1 \approx t2 \implies nabla \vdash swap\ pi2\ t1 \approx t2$
 $\langle proof \rangle$

lemma *ds-empty-equiv-2*:

assumes $ds\ pi1\ pi2 = \{\}$
shows $nabla \vdash t1 \approx swap\ pi1\ t2 \implies nabla \vdash t1 \approx swap\ pi2\ t2$
 $\langle proof \rangle$

lemma *equ-equivariance*:

assumes $nabla \vdash t1 \approx t2$
shows $nabla \vdash swap\ pi\ t1 \approx swap\ pi\ t2$
 $\langle proof \rangle$

lemma *swap-inv-side*:

shows $nabla \vdash swap\ pi\ t1 \approx t2 = nabla \vdash t1 \approx swap\ (rev\ pi)\ t2$
 $\langle proof \rangle$

lemma *equ-swap-abba*:
assumes $n = \text{depth } t1$
shows $nabla \vdash \text{swap } [(a,b)] t1 \approx t2 \implies nabla \vdash t1 \approx \text{swap } [(b,a)] t2$
 $\langle \text{proof} \rangle$

lemma *equ-equiv-pi*:
assumes $\forall a \in ds \ pi1 \ pi2. nabla \vdash a \# t$
shows $nabla \vdash \text{swap } pi1 t \approx \text{swap } pi2 t$
 $\langle \text{proof} \rangle$

lemma *equ-symm*:
shows $(nabla \vdash t1 \approx t2) \implies (nabla \vdash t2 \approx t1)$
 $\langle \text{proof} \rangle$

lemma *equ-trans*:
assumes $nabla \vdash t1 \approx t2 \ nabla \vdash t2 \approx t3$
shows $nabla \vdash t1 \approx t3$
 $\langle \text{proof} \rangle$

lemma *pi-right-equ-help*:
assumes $(n = \text{depth } t)$
shows $nabla \vdash t \approx \text{swap } pi t \implies \forall a \in ds \ [] \ pi. nabla \vdash a \# t$
 $\langle \text{proof} \rangle$

8 Equality

Proves various facts about the equivalence relation.

lemma *equ-refl*:
shows $nabla \vdash t \approx t$
 $\langle \text{proof} \rangle$

lemma *equ-dec-pi*:
assumes $nabla \vdash \text{swap } pi t1 \approx \text{swap } pi t2$
shows $nabla \vdash t1 \approx t2$
 $\langle \text{proof} \rangle$

lemma *equ-involutive-left*:
shows $nabla \vdash \text{swap } (\text{rev } pi) (\text{swap } pi t1) \approx t2 = nabla \vdash t1 \approx t2$
 $\langle \text{proof} \rangle$

lemma *equ-pi-to-left*:
shows $nabla \vdash \text{swap } (\text{rev } pi) t1 \approx t2 = nabla \vdash t1 \approx \text{swap } pi t2$
 $\langle \text{proof} \rangle$

lemma *equ-pi-to-right*:
shows $nabla \vdash t1 \approx \text{swap } (\text{rev } pi) t2 = nabla \vdash \text{swap } pi t1 \approx t2$

<proof>

lemma *equ-involutive-right*:

shows $nabla \vdash t1 \approx \text{swap } (\text{rev } \pi) (\text{swap } \pi t2) = nabla \vdash t1 \approx t2$
<proof>

lemma *equ-pi1-pi2-add*:

assumes $\forall a \in ds \ \pi1 \ \pi2. \ nabla \vdash a \# t$
shows $nabla \vdash \text{swap } \pi1 \ t \approx \text{swap } \pi2 \ t$
<proof>

lemma *pi-right-equ*:

assumes $nabla \vdash t \approx \text{swap } \pi \ t$
shows $\forall a \in ds \ \square \ \pi. \ nabla \vdash a \# t$
<proof>

lemma *equ-pi1-pi2-dec*:

assumes $nabla \vdash \text{swap } \pi1 \ t \approx \text{swap } \pi2 \ t$
shows $\forall a \in ds \ \pi1 \ \pi2. \ nabla \vdash a \# t$
<proof>

lemma *equ-weak*:

assumes $nabla1 \vdash t1 \approx t2$
shows $(nabla1 \cup nabla2) \vdash t1 \approx t2$
<proof>

No term can be equal to one of its proper subterm.

lemma *psub-trm-not-equ*:

shows $\forall t2 \in \text{psub-trms } t1. \ (\neg(\exists \pi. (nabla \vdash t1 \approx \text{swap } \pi t2)))$
<proof>

9 Substitutions

Defines substitutions and composition of substitutions, and establishes some facts of substitution and the equivalence relation.

type-synonym *subst* = (*string* \times *trm*) *list*

fun *look-up* :: *string* \Rightarrow *subst* \Rightarrow *trm* **where**

look-up *X* \square = *Susp* \square *X* |

look-up *X* (*x*#*xs*) = (*if* (*X* = *fst* *x*) *then* (*snd* *x*) *else* *look-up* *X* *xs*)

fun *subst* :: *subst* \Rightarrow *trm* \Rightarrow *trm* **where**

subst-unit: *subst* *s* (*Unit*) = *Unit* |

subst-susp: *subst* *s* (*Susp* π *X*) = *swap* π (*look-up* *X* *s*) |

subst-atom: *subst* *s* (*Atom* *a*) = *Atom* *a* |

subst-abst: *subst* *s* (*Abst* *a* *t*) = *Abst* *a* (*subst* *s* *t*) |

subst-paar: *subst* *s* (*Paar* *t1* *t2*) = *Paar* (*subst* *s* *t1*) (*subst* *s* *t2*) |

subst-func: *subst* *s* (*Func* *F* *t*) = *Func* *F* (*subst* *s* *t*)

declare *subst-susp* [*simp del*]

The notion of composition of substitutions (adapted from Martin Coen's Classical Computational Logic).

fun *alist-rec* :: *substs* \Rightarrow *substs* \Rightarrow (*string* \Rightarrow *trm* \Rightarrow *substs* \Rightarrow *substs* \Rightarrow *substs*) \Rightarrow *substs*
where

alist-rec [] *c d* = *c* |
alist-rec (*p* # *al*) *c d* = *d* (*fst p*) (*snd p*) *al* (*alist-rec al c d*)

definition *comp* :: *substs* \Rightarrow *substs* \Rightarrow *substs* (**infixl** $\langle \cdot \rangle$ 81) **where**
s1 \cdot *s2* \equiv *alist-rec s2 s1* ($\lambda x y xs g. (x,subst s1 y) \# g$)

The domain of substitutions.

definition *domn* :: (*trm* \Rightarrow *trm*) \Rightarrow *string set* **where**
domn s \equiv {*X*. (*s* (*Susp* [] *X*)) \neq (*Susp* [] *X*)}

substitutions extend freshness environments.

definition *ext-subst* :: *fresh-envs* \Rightarrow (*trm* \Rightarrow *trm*) \Rightarrow *fresh-envs* \Rightarrow *bool* (- \models - -
[80,80,80] 80) **where**
nabla1 \models *s* *nabla2* \equiv ($\forall (a,X) \in nabla2. nabla1 \vdash a \# s$ (*Susp* [] *X*))

Alpha-equality for substitutions

definition *subst-equ* :: *fresh-envs* \Rightarrow (*trm* \Rightarrow *trm*) \Rightarrow (*trm* \Rightarrow *trm*) \Rightarrow *bool* (- \models - \approx
- [80,80,80] 80)
where
nabla \models *s1* \approx *s2* \equiv $\forall X \in (domn s1 \cup domn s2). (nabla \vdash s1$ (*Susp* [] *X*) \approx *s2*
(*Susp* [] *X*))

lemma *subst-equ-symm*:

assumes *nabla* \models *s1* \approx *s2*
shows *nabla* \models *s2* \approx *s1*
 \langle *proof* \rangle

lemma *subst-equ-refl*:

shows *nabla* \models *s* \approx *s*
 \langle *proof* \rangle

lemma *not-in-domn*:

assumes *X* \notin (*domn s*)
shows *s* (*Susp* [] *X*) = (*Susp* [] *X*)
 \langle *proof* \rangle

lemma *subst-swap-comm*:

shows *subst s* (*swap pi t*) = *swap pi* (*subst s t*)
 \langle *proof* \rangle

lemma *subst-not-occurs*:

assumes $\neg(\text{occurs } X \ t)$
shows $\text{subst } [(X, t2)] \ t = t$
 $\langle \text{proof} \rangle$

lemma *id-subst* [*simp*]:
shows $\text{subst } [] \ t = t$
 $\langle \text{proof} \rangle$

lemma *subst-comp-id* [*simp*]:
shows $\text{subst } (s \cdot []) = \text{subst } s$
 $\langle \text{proof} \rangle$

lemma *id-comp-subst* [*simp*]:
shows $\text{subst } ([] \cdot s) = \text{subst } s$
 $\langle \text{proof} \rangle$

lemma *subst-comp-expand*:
shows $\text{subst } (s1 \cdot s2) \ t = \text{subst } s1 \ (\text{subst } s2 \ t)$
 $\langle \text{proof} \rangle$

lemma *subst-assoc*:
shows $\text{subst } (s1 \cdot (s2 \cdot s3)) = \text{subst } ((s1 \cdot s2) \cdot s3)$
 $\langle \text{proof} \rangle$

lemma *fresh-subst*:
assumes $\text{nabla} a1 \vdash a \ \# \ t$
and $\text{nabla} a2 \models (\text{subst } s) \ \text{nabla} a1$
shows $\text{nabla} a2 \vdash a \ \# \ \text{subst } s \ t$
 $\langle \text{proof} \rangle$

lemma *equ-subst*:
assumes $\text{nabla} a1 \vdash t1 \approx t2$
and $\text{nabla} a2 \models (\text{subst } s) \ \text{nabla} a1$
shows $\text{nabla} a2 \vdash (\text{subst } s \ t1) \approx (\text{subst } s \ t2)$
 $\langle \text{proof} \rangle$

lemma *unif-1*:
assumes $\text{nabla} a \vdash \text{subst } s \ (\text{Susp } \pi \ X) \approx \text{subst } s \ t$
shows $\text{nabla} a \models \text{subst } (s \cdot [(X, \text{swap } (\text{rev } \pi) \ t)]) \approx \text{subst } s$
 $\langle \text{proof} \rangle$

lemma *subst-equ-a*:
assumes $\text{nabla} a \models (\text{subst } s1) \approx (\text{subst } s2)$
and $\text{nabla} a \vdash (\text{subst } s2 \ t1) \approx t2$
shows $\text{nabla} a \vdash (\text{subst } s1 \ t1) \approx t2$
 $\langle \text{proof} \rangle$

lemma *unif-2a*:
assumes $\text{nabla} a \models \text{subst } s1 \approx \text{subst } s2$

and ($nabla \vdash subst\ s2\ t1 \approx subst\ s2\ t2$)
shows ($nabla \vdash subst\ s1\ t1 \approx subst\ s1\ t2$)
 $\langle proof \rangle$

lemma *unif-2b*:
assumes $nabla \models subst\ s1 \approx subst\ s2$
and $nabla \vdash a \# subst\ s2\ t$
shows $nabla \vdash a \# subst\ s1\ t$
 $\langle proof \rangle$

lemma *subst-equ-to-trm*:
assumes $nabla \models subst\ s1 \approx subst\ s2$
shows $nabla \vdash subst\ s1\ t \approx subst\ s2\ t$
 $\langle proof \rangle$

lemma *subst-cancel-right*:
assumes $nabla \models (subst\ s1) \approx (subst\ s2)$
shows $nabla \models (subst\ (s1 \cdot s)) \approx (subst\ (s2 \cdot s))$
 $\langle proof \rangle$

lemma *subst-trans*:
assumes $nabla \models subst\ s1 \approx subst\ s2$ $nabla \models subst\ s2 \approx subst\ s3$
shows $nabla \models subst\ s1 \approx subst\ s3$
 $\langle proof \rangle$

If occurs holds, then one subterm is equal to (subst s (Susp pi X))

lemma *occurs-sub-trm-equ*:
assumes *occurs* $X\ t1$
shows $\exists t2 \in sub-trms\ (subst\ s\ t1). (\exists pi. (nabla \vdash subst\ s\ (Susp\ pi\ X) \approx (swap\ pi\ t2)))$
 $\langle proof \rangle$

lemma *ext-subst-strong*:
assumes $nabla1 \models (subst\ s)\ (nabla2 \cup nabla3)$
shows $nabla1 \models (subst\ s)\ nabla2$ **and** $nabla1 \models (subst\ s)\ nabla3$
 $\langle proof \rangle$

lemma *ext-subst-id*:
shows $nabla \models (subst\ [])\ nabla$
 $\langle proof \rangle$

10 Most General Unifiers

Defines the notion of unification problems and reduction rules over sets of such problems; proves that every reduction leading to the empty set produces an mgu.

syntax *equ-prob* :: $trm \Rightarrow trm \Rightarrow (trm \times trm)$ (**infix** $\approx?$ 81)
fresh-prob :: $string \Rightarrow trm \Rightarrow (string \times trm)$ (**infix** $\#?$ 81)

translations $t1 \approx? t2 \rightarrow (t1, t2)$ $a \#? t \rightarrow (a, t)$

All solutions for a unification problem.

type-synonym $problem\text{-}type = ((trm \times trm) list) \times ((string \times trm) list)$ **type-synonym** $unifier\text{-}type = fresh\text{-}envs \times substs$ **definition** $U :: problem\text{-}type \Rightarrow (unifier\text{-}type set)$ **where** $all\text{-}solutions\text{-}def:$
$$U P \equiv \{(nabla, s). \\ (\forall (t1, t2) \in set (fst P). nabla \vdash subst s t1 \approx subst s t2) \wedge \\ (\forall (a, t) \in set (snd P). nabla \vdash a \# subst s t)\}$$

The set of variables in unification problems.

type-synonym $epr obs = ((trm \times trm) list)$ **type-synonym** $fpr obs = ((string \times trm) list)$ **type-synonym** $pr obs = epr obs \times fpr obs$ **fun** $vars\text{-}fpr obs :: ((string \times trm) list) \Rightarrow (string set)$ **where** $vars\text{-}fpr obs [] = \{\}$ $vars\text{-}fpr obs (x\#xs) = (vars\text{-}trm (snd x)) \cup (vars\text{-}fpr obs xs)$ **fun** $vars\text{-}epr obs :: ((trm \times trm) list) \Rightarrow (string set)$ **where** $vars\text{-}epr obs [] = \{\}$ $vars\text{-}epr obs (x\#xs) = (vars\text{-}trm (snd x)) \cup (vars\text{-}trm (fst x)) \cup (vars\text{-}epr obs xs)$ **definition** $vars\text{-}pr obs :: problem\text{-}type \Rightarrow nat$ **where** $vars\text{-}pr obs P \equiv card((vars\text{-}fpr obs (snd P)) \cup (vars\text{-}epr obs (fst P)))$

Most general unifier

definition $mgu :: problem\text{-}type \Rightarrow unifier\text{-}type \Rightarrow bool$ **where** $mgu P unif \equiv$
$$\forall (nabla, s1) \in U P. (\exists s2. (nabla \models (subst s2) (fst unif)) \wedge \\ (nabla \models subst (s2 \cdot (snd unif)) \approx subst s1))$$

Idempotency of a unifier

definition $idem :: unifier\text{-}type \Rightarrow bool$ **where** $idem unif \equiv (fst unif) \models subst ((snd unif) \cdot (snd unif)) \approx subst (snd unif)$

Application of a substitution to a problem

definition $apply\text{-}subst\text{-}epr obs :: substs \Rightarrow epr obs \Rightarrow epr obs$ **where** $apply\text{-}subst\text{-}epr obs s P \equiv map (\lambda(t1, t2). (subst s t1 \approx? subst s t2)) P$ **definition** $apply\text{-}subst\text{-}fpr obs :: substs \Rightarrow fpr obs \Rightarrow fpr obs$

where $apply\text{-subst}\text{-fprobs } s P \equiv map (\lambda(a, t). (a \#? subst s t)) P$

definition $apply\text{-subst} :: substs \Rightarrow problem\text{-type} \Rightarrow problem\text{-type}$

where $apply\text{-subst } s P \equiv (map (\lambda(t1, t2). (subst s t1 \approx? subst s t2))) (fst P),$
 $map (\lambda(a, t). (a \#? (subst s t))) (snd P)$

Equality reductions

inductive $s\text{-red} :: problem\text{-type} \Rightarrow substs \Rightarrow problem\text{-type} \Rightarrow bool \ (- \vdash - \rightsquigarrow -$
 $[80,80,80] 80)$

where

$unit\text{-sred}[intro!]: ((Unit \approx? Unit) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys) \mid$

$paar\text{-sred}[intro!]: ((Paar t1 t2 \approx? Paar s1 s2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? s1) \# (t2 \approx? s2) \# xs, ys)$

$func\text{-sred}[intro!]: ((Func F t1 \approx? Func F t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys) \mid$

$abst\text{-aa}\text{-sred}[intro!]: ((Abst a t1 \approx? Abst a t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys) \mid$

$abst\text{-ab}\text{-sred}[intro!]: a \neq b \implies$

$((Abst a t1 \approx? Abst b t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? swap [(a, b)$

$t2] \# xs, (a \#? t2) \# ys) \mid$

$atom\text{-sred}[intro!]: ((Atom a \approx? Atom a) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys) \mid$

$susp\text{-sred}[intro!]: ((Susp pi1 X \approx? Susp pi2 X) \# xs, ys)$

$\vdash [] \rightsquigarrow (xs, (map (\lambda a. a \#? Susp [] X) (ds\text{-list } pi1 pi2)) @ ys) \mid$

$var\text{-1}\text{-sred}[intro!]: \neg(occurs X t) \implies ((Susp pi X \approx? t) \# xs, ys)$

$\vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-subst} [(X, swap (rev pi) t)]$

$(xs, ys) \mid$

$var\text{-2}\text{-sred}[intro!]: \neg(occurs X t) \implies ((t \approx? Susp pi X) \# xs, ys)$

$\vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-subst} [(X, swap (rev pi) t)]$

(xs, ys)

inductive-cases $s\text{-red}\text{-elims}$:

$((Unit \approx? Unit) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$

$((Paar t1 t2 \approx? Paar s1 s2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? s1) \# (t2 \approx? s2) \# xs, ys)$

$((Func F t1 \approx? Func F t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys)$

$((Abst a t1 \approx? Abst a t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys)$

$((Abst a t1 \approx? Abst b t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? swap [(a, b)] t2) \# xs, (a \#? t2) \# ys)$

$((Atom a \approx? Atom a) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$

$((Susp pi1 X \approx? Susp pi2 X) \# xs, ys) \vdash [] \rightsquigarrow (xs, (map (\lambda a. a \#? Susp [] X) (ds\text{-list } pi1 pi2)) @ ys)$

$((Susp pi X \approx? t) \# xs, ys) \vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-subst} [(X, swap (rev pi) t)]$

(xs, ys)

$((t \approx? Susp pi X) \# xs, ys) \vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-subst} [(X, swap (rev pi) t)]$

(xs, ys)

lemma $sred\text{-symm}$:

assumes $((t1 \approx? t2) \# xs, ys) \vdash s \rightsquigarrow P2$

shows $\exists P3. ((t2 \approx? t1) \# xs, ys) \vdash s \rightsquigarrow P3$

$\langle proof \rangle$

Weakening of freshness

lemma *solution-weak*:
assumes $(nabla1, s) \in UP$
shows $(nabla1 \cup nabla3, s) \in UP$
 $\langle proof \rangle$

lemma *solution-comp-id*:
shows $((nabla, s) \in UP) = ((nabla, s \cdot []) \in UP)$ **and**
 $((nabla, s) \in UP) = ((nabla, [] \cdot s) \in UP)$
 $\langle proof \rangle$

lemma *solutions-subst-assoc*:
 $((nabla, s1 \cdot (s2 \cdot s3)) \in UP) = ((nabla, (s1 \cdot s2) \cdot s3) \in UP)$
 $\langle proof \rangle$

Freshness reductions

inductive *c-red* :: *problem-type* \Rightarrow *fresh-envs* \Rightarrow *problem-type* \Rightarrow *bool* ($- \vdash - \rightarrow -$
 $[80,80,80] 80$)
where
unit-cred[intro!]: $([], (a \#? Unit) \# xs) \vdash \{\} \rightarrow ([], xs) \mid$
paar-cred[intro!]: $([], (a \#? Paar t1 t2) \# xs) \vdash \{\} \rightarrow ([], (a \#? t1) \# (a \#? t2) \# xs) \mid$
func-cred[intro!]: $([], (a \#? Func F t) \# xs) \vdash \{\} \rightarrow ([], (a \#? t) \# xs) \mid$
abst-aa-cred[intro!]: $([], (a \#? Abst a t) \# xs) \vdash \{\} \rightarrow ([], xs) \mid$
abst-ab-cred[intro!]: $a \neq b \implies ([], (a \#? Abst b t) \# xs) \vdash \{\} \rightarrow ([], (a \#? t) \# xs) \mid$
atom-cred[intro!]: $a \neq b \implies ([], (a \#? Atom b) \# xs) \vdash \{\} \rightarrow ([], xs) \mid$
susp-cred[intro!]: $([], (a \#? Susp pi X) \# xs) \vdash \{(swapas (rev pi) a), X\} \rightarrow ([], xs)$

It only reduces the freshness part after the equations list is empty

lemma *c-red-eqs-empty*:
assumes $P1 \vdash s \rightarrow P2$
shows $fst P1 = []$
 $\langle proof \rangle$

Unification reduction sequences

inductive *red-plus* :: *problem-type* \Rightarrow *unifier-type* \Rightarrow *problem-type* \Rightarrow *bool* ($- \models -$
 $\Rightarrow - [80,80,80] 80$)
where
sred-single[intro!]: $\llbracket P1 \vdash s1 \rightsquigarrow P2 \rrbracket \implies P1 \models (\{\}, s1) \Rightarrow P2 \mid$
cred-single[intro!]: $\llbracket P1 \vdash nabla1 \rightarrow P2 \rrbracket \implies P1 \models (nabla1, []) \Rightarrow P2 \mid$
sred-step[intro!]: $\llbracket P1 \vdash s1 \rightsquigarrow P2; P2 \models (nabla2, s2) \Rightarrow P3 \rrbracket \implies P1 \models (nabla2, (s2 \cdot s1)) \Rightarrow P3$
 \mid
cred-step[intro!]: $\llbracket P1 \vdash nabla1 \rightarrow P2; P2 \models (nabla2, []) \Rightarrow P3 \rrbracket \implies P1 \models (nabla2 \cup nabla1, []) \Rightarrow P3$

Symmetry of the reduction

lemma *red-plus-symm*:
assumes $P1 = ((t1 \approx? t2) \# xs, ys)$
and $P1 \models (nabla, s) \Rightarrow P2$
shows $\exists nabla1 s1 P3. ((t2 \approx? t1) \# xs, ys) \models (nabla1, s1) \Rightarrow P3$
 $\langle proof \rangle$

lemma *mgu-idem*:

assumes $(nabla1, s1) \in U P$
and $\forall (nabla2, s2) \in U P. nabla2 \models (subst\ s2)\ nabla1 \wedge nabla2 \models subst(s2 \cdot s1) \approx_{subst\ s2}$
shows $mgu\ P\ (nabla1, s1) \wedge idem\ (nabla1, s1)$
<proof>

lemma *problem-subst-comm*:

shows $((nabla, s2) \in U (apply\ subst\ s1\ P)) = ((nabla, (s2 \cdot s1)) \in U P)$
<proof>

Preservation of solutions

lemma *P1-to-P2-sred*:

assumes $(nabla1, s1) \in U P1$ **and** $P1 \vdash_{s2} \rightsquigarrow P2$
shows $(nabla1, s1) \in U P2 \wedge (nabla1 \models_{subst} (s1 \cdot s2) \approx_{subst} s1)$
<proof>

Auxiliary lemma for completeness

lemma *P1-from-P2-sred*:

assumes $(nabla1, s1) \in U P2$ **and** $P1 \vdash_{s2} \rightsquigarrow P2$
shows $(nabla1, s1 \cdot s2) \in U P1$
<proof>

lemma *P1-to-P2-cred*:

assumes $(nabla1, s1) \in U P1$
and $P1 \vdash nabla2 \rightarrow P2$
shows $(nabla1, s1) \in U P2 \wedge (nabla1 \models (subst\ s1)\ nabla2)$
<proof>

lemma *P1-from-P2-cred*:

assumes $(nabla1, s1) \in U P2$
and $P1 \vdash nabla2 \rightarrow P2$ **and** $nabla3 \models (subst\ s1)\ nabla2$
shows $(nabla1 \cup nabla3, s1) \in U P1$
<proof>

lemma *P1-to-P2-red-plus1*:

assumes $P1 \models (nabla, s) \Rightarrow P2$
and $(nabla1, s1) \in U P1$
shows $(nabla1, s1) \in U P2$
<proof>

lemma *P1-to-P2-red-plus3*:

assumes $P1 \models (nabla, s) \Rightarrow P2$
and $(nabla1, s1) \in U P1$
shows $nabla1 \models (subst\ s1)\ nabla$
<proof>

lemma *P1-to-P2-red-plus2*:
assumes $P1 \models (nabla, s) \Rightarrow P2$
and $(nabla1, s1) \in U P1$
shows $nabla1 \models subst (s1 \cdot s) \approx subst s1$
 $\langle proof \rangle$

lemma *P1-from-P2-red-plus*:
assumes $P1 \models (nabla, s) \Rightarrow P2$
and $(nabla1, s1) \in U P2$ **and** $nabla3 \models (subst s1)(nabla)$
shows $(nabla1 \cup nabla3, (s1 \cdot s)) \in U P1$
 $\langle proof \rangle$

lemma *mgu*:
assumes $P \models (nabla, s) \Rightarrow ([], [])$
shows $mgu P (nabla, s) \wedge idem (nabla, s)$
 $\langle proof \rangle$

11 Termination

Defines a lexicographic termination measure and proves that all the unification reductions decrease this measure.

lemma *apply-subst-equivalence*:
shows $apply\text{-}subst\ s\ P = (apply\text{-}subst\text{-}eprobs\ s\ (fst\ P), apply\text{-}subst\text{-}fprobs\ s\ (snd\ P))$
 $\langle proof \rangle$

fun *size-trm* :: $trm \Rightarrow nat$
where
 $size\text{-}trm\ (Unit) = 1 \mid$
 $size\text{-}trm\ (Atom\ a) = 1 \mid$
 $size\text{-}trm\ (Susp\ pi\ X) = 1 \mid$
 $size\text{-}trm\ (Abst\ a\ t) = 1 + size\text{-}trm\ t \mid$
 $size\text{-}trm\ (Func\ F\ t) = 1 + size\text{-}trm\ t \mid$
 $size\text{-}trm\ (Paar\ t\ t') = 1 + (size\text{-}trm\ t) + (size\text{-}trm\ t')$

fun *size-fprobs* :: $fprobs \Rightarrow nat$
where
 $size\text{-}fprobs\ [] = 0 \mid$
 $size\text{-}fprobs\ (x\#\ xs) = (size\text{-}trm\ (snd\ x)) + (size\text{-}fprobs\ xs)$

fun *size-eprobs* :: $eprobs \Rightarrow nat$
where
 $size\text{-}eprobs\ [] = 0 \mid$
 $size\text{-}eprobs\ (x\#\ xs) = (size\text{-}trm\ (fst\ x)) + (size\text{-}trm\ (snd\ x)) + (size\text{-}eprobs\ xs)$

lemma *size-swap* [*simp*]: $size\text{-}trm\ (swap\ pi\ t) = size\text{-}trm\ t$

<proof>

definition *rank-r*

where

rank-r = *measures* [$\lambda(\text{eprobs}, \text{fprobs}). \text{card} (\text{vars-eprobs eprobs}),$
 $\lambda(\text{eprobs}, \text{fprobs}). \text{size-eprobs eprobs},$
 $\lambda(\text{eprobs}, \text{fprobs}). \text{size-fprobs fprobs}$]

lemma *vars-term-finite [simp]: finite (vars-trm t)*

<proof>

lemma *vars-eprobs-finite [simp]: finite (vars-eprobs P)*

<proof>

lemma *not-occurs-trm: $\neg \text{occurs } X t \longrightarrow X \notin \text{vars-trm } t$*

<proof>

lemma *not-occurs-subst: $\neg \text{occurs } X t1 \longrightarrow X \notin \text{vars-trm} (\text{subst} [(X, \text{swap } pi2 t1)] t2)$*

<proof>

lemma *not-occurs-list: $\neg \text{occurs } X t \longrightarrow$*

$X \notin \text{vars-eprobs} (\text{apply-subst-eprobs} [(X, \text{swap } pi t)] xs)$

<proof>

lemma *vars-equ:*

assumes $\neg \text{occurs } X t1$ **and** $\text{occurs } X t2$

shows $\text{vars-trm} (\text{subst} [(X, \text{swap } pi t1)] t2) = (\text{vars-trm } t1 \cup \text{vars-trm } t2) - \{X\}$

<proof>

lemma *vars-subseteq:*

assumes $\neg \text{occurs } X t$

shows $\text{vars-eprobs} (\text{apply-subst-eprobs} [(X, \text{swap } pi t)] xs) \subseteq (\text{vars-trm } t \cup \text{vars-eprobs } xs)$

<proof>

lemma *vars-decrease:*

assumes $\neg \text{occurs } X t$

shows $\text{card} (\text{vars-eprobs} (\text{apply-subst-eprobs} [(X, \text{swap } pi t)] xs))$

$< \text{card} (\text{insert } X (\text{vars-trm } t \cup \text{vars-eprobs } xs))$

<proof>

lemma *rank-r-cred*:
assumes $P1 \vdash (nabla) \rightarrow P2$
shows $(P2, P1) \in \text{rank-r}$
 $\langle \text{proof} \rangle$

lemma *rank-r-sred*:
assumes $P1 \vdash s \rightsquigarrow P2$
shows $(P2, P1) \in \text{rank-r}$
 $\langle \text{proof} \rangle$

lemma *rank-r-trans*: $\llbracket (P1, P2) \in \text{rank-r}; (P2, P3) \in \text{rank-r} \rrbracket \Longrightarrow (P1, P3) \in \text{rank-r}$
 $\langle \text{proof} \rangle$

lemma *rank-r-red-plus*:
assumes $P1 \models (s, nabla) \Rightarrow P2$
shows $(P2, P1) \in \text{rank-r}$
 $\langle \text{proof} \rangle$

lemma *wf-rank-r*:
shows *wf* (*rank-r*)
 $\langle \text{proof} \rangle$

12 Unification

Proves that all problems that are stuck and fail, have no unifier.

definition *stuck* :: *problem-type set where*
stuck-def: $\text{stuck} \equiv \{ P1. \neg(\exists P2 \text{ nabla } s. P1 \models (nabla, s) \Rightarrow P2) \}$

inductive *fail* :: *problem-type \Rightarrow bool where*
fail-occur-abst [*intro!*]: $\llbracket \text{occurs } X \ t \rrbracket \Longrightarrow \text{fail } ((\text{Susp } \text{pi } X \approx ? \text{Abst } a \ t) \# \text{xs}, \text{ys}) \mid$
fail-occur-func [*intro!*]: $\llbracket \text{occurs } X \ t \rrbracket \Longrightarrow \text{fail } (\text{Susp } \text{pi } X \approx ? \text{Func } F \ t \# \text{xs}, \text{ys}) \mid$
fail-occur-paar [*intro!*]: $\llbracket \text{occurs } X \ t1 \vee \text{occurs } X \ t2 \rrbracket \Longrightarrow \text{fail } (\text{Susp } \text{pi } X \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}) \mid$
fail-fresh-atom [*intro!*]: $\text{fail } ([], a \# ? \text{Atom } a \# \text{ys}) \mid$
fail-diff-atoms [*intro!*]: $a \neq b \Longrightarrow \text{fail } (\text{Atom } a \approx ? \text{Atom } b \# \text{xs}, \text{ys}) \mid$
fail-abst-unit [*intro!*]: $\text{fail } (\text{Abst } a \approx ? \text{Unit} \# \text{xs}, \text{ys}) \mid$
fail-abst-atom [*intro!*]: $\text{fail } (\text{Abst } a \approx ? \text{Atom } b \# \text{xs}, \text{ys}) \mid$
fail-abst-paar [*intro!*]: $\text{fail } (\text{Abst } a \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}) \mid$
fail-func-abst [*intro!*]: $\text{fail } (\text{Func } F \ t1 \approx ? \text{Abst } a \ t \# \text{xs}, \text{ys}) \mid$
fail-atom-unit [*intro!*]: $\text{fail } (\text{Atom } b \approx ? \text{Unit} \# \text{xs}, \text{ys}) \mid$
fail-paar-unit [*intro!*]: $\text{fail } (\text{Paar } t1 \ t2 \approx ? \text{Unit} \# \text{xs}, \text{ys}) \mid$
fail-func-unit [*intro!*]: $\text{fail } (\text{Func } F \ t1 \approx ? \text{Unit} \# \text{xs}, \text{ys}) \mid$
fail-atom-paar [*intro!*]: $\text{fail } (\text{Atom } a \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}) \mid$
fail-func-atom [*intro!*]: $\text{fail } (\text{Func } F \ t1 \approx ? \text{Atom } a \# \text{xs}, \text{ys}) \mid$
fail-func-paar [*intro!*]: $\text{fail } (\text{Func } F \ t \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}) \mid$
fail-diff-func [*intro!*]: $\llbracket F1 \neq F2 \rrbracket \Longrightarrow \text{fail } (\text{Func } F1 \ t1 \approx ? \text{Func } F2 \ t2 \# \text{xs}, \text{ys}) \mid$
fail-sym [*intro!*]: $\text{fail } (s \approx ? t \# \text{xs}, \text{ys}) \Longrightarrow \text{fail } (t \approx ? s \# \text{xs}, \text{ys})$

definition

normal-form :: *problem-type* \Rightarrow *problem-type set* **where**
normal-form $P1 \equiv$ if $P1 \in stuck$ then $\{P1\}$ else $\{P2. \exists nabl a s. P1 \models (nabl a, s) \Rightarrow P2 \wedge P2 \in stuck\}$

lemma *U-equ-symm:*

shows $U (s \approx ? t \# xs, ys) = U (t \approx ? s \# xs, ys)$
 $\langle proof \rangle$

lemma *fail-then-empty:*

assumes *fail* $P1$
shows $U P1 = \{\}$
 $\langle proof \rangle$

lemma *not-reduce-then-fail:*

assumes $\neg (\exists nabl a s P'. ((t1 \approx ? t2) \# xs, ys) \models (nabl a, s) \Rightarrow P')$
shows *fail* $((t1 \approx ? t2) \# xs, ys)$
 $\langle proof \rangle$

lemma *fresh-reduces-if-not-atom:*

assumes $t \neq Atom a$
shows $\exists P2 nabl a s. (\[], (a \# ? t) \# xs) \models (nabl a, s) \Rightarrow P2$
 $\langle proof \rangle$

lemma *empty-stuck:*

shows $(\[], \[]) \in stuck$
 $\langle proof \rangle$

lemma *fail-is-stuck:*

assumes *fail* P
shows $P \in stuck$
 $\langle proof \rangle$

lemma *stuck-equiv*:
shows $stuck = \{(\square, \square)\} \cup \{P1. fail P1\}$
 $\langle proof \rangle$

lemma *u-empty-sred*:
assumes $P1 \vdash s \rightsquigarrow P2$ **and** $U P2 = \{\}$
shows $U P1 = \{\}$
 $\langle proof \rangle$

lemma *u-empty-cred*:
assumes $P1 \vdash nabla \rightarrow P2$ **and** $U P2 = \{\}$
shows $U P1 = \{\}$
 $\langle proof \rangle$

lemma *u-empty-red-plus*:
assumes $P1 \models (nabla, s) \Rightarrow P2$ **and** $U P2 = \{\}$
shows $U P1 = \{\}$
 $\langle proof \rangle$

lemma *empty-then-fail*:
assumes $U P1 = \{\}$
shows $(\forall P \in normal\text{-form } P1. fail P)$
 $\langle proof \rangle$

lemma *not-empty-then-not-fail*:
assumes $U P1 \neq \{\}$
shows $\neg(\exists P \in normal\text{-form } P1. fail P)$
 $\langle proof \rangle$

References

- [1] M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, D. Nantes-Sobrinho, and A. C. R. Oliveira. A formalisation of nominal α -equivalence with a, c, and AC function symbols. *Theor. Comput. Sci.*, 781:3–23, 2019.
- [2] M. Ayala-Rincón, M. Fernández, and A. C. R. Oliveira. Completeness

- in PVS of a nominal unification algorithm. In M. R. F. Benevides and R. Thiemann, editors, *Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2015, Natal, Brazil, August 31 - September 1, 2015*, volume 323 of *Electronic Notes in Theoretical Computer Science*, pages 57–74. Elsevier, 2015.
- [3] W. L. R. de Carvalho. *Nominal Equational Problems Modulo Associativity, Commutativity and Associativity-Commutativity*. PhD thesis, Graduate Program in Informatics, University of Brasília, 2019. in English.
- [4] A. C. R. Oliveira. *Unification, Confluence, and Intersection Types for Nominal Rewriting Systems*. PhD thesis, Graduate Program in Informatics, University of Brasília, 2016. in English.
- [5] C. Urban. Nominal unification revisited. In M. Fernández, editor, *Proceedings 24th International Workshop on Unification, UNIF 2010, Edinburgh, United Kingdom, 14th July 2010*, volume 42 of *EPTCS*, pages 1–11, 2010.
- [6] C. Urban, A. M. Pitts, and M. Gabbay. Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497, 2004.