

# Nominal Unification

Guilherme Borges Brandão<sup>1</sup>      Thomas Ammer<sup>2</sup>  
Daniele Nantes Sobrinho<sup>1</sup>      Mauricio Ayala-Rinción<sup>1</sup>  
Christian Urban<sup>2</sup>      Maribel Fernández<sup>2</sup>  
Mohammad Abdulaziz<sup>2</sup>

<sup>1</sup> Universidade de Brasília, Brasília D.F., Brazil

<sup>2</sup> King's College London, London, U.K.

April 24, 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

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Swappings of Pairs of Atoms</b>	<b>2</b>
<b>3</b>	<b>Terms</b>	<b>3</b>
<b>4</b>	<b>Facts about Atoms Occurring in Swappings</b>	<b>7</b>

<b>5</b>	<b>Disagreement Sets</b>	<b>8</b>
<b>6</b>	<b>Freshness</b>	<b>11</b>
<b>7</b>	<b>Pre-Equality</b>	<b>13</b>
<b>8</b>	<b>Equality</b>	<b>28</b>
<b>9</b>	<b>Substitutions</b>	<b>31</b>
<b>10</b>	<b>Most General Unifiers</b>	<b>39</b>
<b>11</b>	<b>Termination</b>	<b>58</b>
<b>12</b>	<b>Unification</b>	<b>66</b>
	<b>References</b>	<b>83</b>

## 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  $\alpha$ -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  $\alpha$ -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
  by simp

lemma swapas-ab-ba:
  shows swapas [(a,b)] = swapas [(b,a)]
  by auto

lemma swapas-append:
  shows swapas (pi1@pi2) a = swapas pi1 (swapas pi2 a)
  by (induct pi1) auto

lemma swapas-inv [simp]:
  shows swapas (rev pi) (swapas pi a) = a
  by (induct-tac pi) (auto simp add: swapas-append)

lemma swapas-rev-pi-a:
  assumes swapas pi a = b
  shows swapas (rev pi) b = a
using assms by auto

lemma swapas-rev-swapas-id [simp]:
  shows swapas pi (swapas (rev pi) a) = a
  by (metis swapas-rev-pi-a)

lemma swapas-rev-pi-b:
  assumes swapas (rev pi) a = b
  shows swapas pi b = a
using assms by auto

lemma swapas-comm:
  shows (swapas (pi@[(a,b)]) c) = (swapas ((swapas pi a,swapas pi b)@pi) c)
  by (metis swapa.simps swapas.simps(1,2) swapas-append swapas-rev-pi-a)

```

### 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**  $swap\ []\ t = t$   
**by** (*induct-tac* *t*) (*auto*)

**lemma** *swap-append*:  
**shows**  $swap\ (pi1@pi2)\ t = swap\ pi1\ (swap\ pi2\ t)$   
**using** *swapas-append* **by** (*induct* *t* *arbitrary: pi1 pi2*) *auto*

**lemma** *swap-empty*:  
**assumes**  $swap\ pi\ t = Susp\ []\ X$   
**shows**  $pi = []$   
**using** *assms* *swap.elims* **by** *blast*

The depth of terms used as measure.

**fun** *depth* :: *trm*  $\Rightarrow$  *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*:  $Suc(max\ x\ y) = z \longrightarrow y < z$   
**by**(*auto*)

**lemma** *swap-depth* [simp]:  
**shows**  $depth\ (swap\ pi\ t) = depth\ t$   
**by** (*induct* *t*) (*auto*)

The occurs predicate and variables inside terms.

**fun** *occurs* :: *string*  $\Rightarrow$  *trm*  $\Rightarrow$  *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  
**by** (induct t) auto

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  
**using** assms  
**by**(induct t2)(auto)

**lemma** t-sub-trms-t:  
**shows** t  $\in$  sub-trms t  
**by** (induct t) (auto)

**lemma** abst-psub-trms:  
**assumes** Abst a t1  $\in$  sub-trms t2  
**shows** t1  $\in$  psub-trms t2  
**using** assms  
**by** (induct t2 arbitrary: t1)  
(auto simp add: t-sub-trms-t intro: psub-sub-trms)

```

lemma func-psub-trms:
  assumes Func F t1 ∈ sub-trms t2
  shows t1 ∈ psub-trms t2
  using assms
  by (induct t2)
    (auto simp add: t-sub-trms-t intro: psub-sub-trms)

lemma paar-psub-trms:
  assumes Paar t1 t2 ∈ sub-trms t3
  shows t1 ∈ psub-trms t3 and t2 ∈ psub-trms t3
  using assms
  by (induct t3) (auto simp add: t-sub-trms-t intro: psub-sub-trms)

lemma t-not-in-psub-trms-t:
  shows t ∉ psub-trms t
proof(induct t)
  case (Abst x1 t)
  then show ?case
    using abst-psub-trms by auto
next
  case (Susp x1 x2)
  then show ?case by simp
next
  case Unit
  then show ?case by simp
next
  case (Atom x)
  then show ?case by simp
next
  case (Paar t1 t2)
  then have Paar t1 t2 ∉ sub-trms t1 Paar t1 t2 ∉ sub-trms t2
    using paar-psub-trms by auto
  then show ?case by simp
next
  case (Func f t)
  then show ?case
    using func-psub-trms by auto
qed

lemma if-sub-not-eq-then-psub:
  assumes t1 ∈ sub-trms t2 t1 ≠ t2
  shows t1 ∈ psub-trms t2
  using assms
  by (induct t2) auto

lemma depth-psub-trms:
  assumes t1 ∈ psub-trms t2
  shows depth t1 < depth t2

```

```

using assms
proof(induct t2 arbitrary: t1)
  case (Abst a t)
  then show ?case
    using depth.simps(4)[of a t]
      psub-trms.simps(5)[of a t] if-sub-not-eq-then-psub
    by fastforce
next
  case (Susp pi X)
  then show ?case by simp
next
  case Unit
  then show ?case by simp
next
  case (Atom a)
  then show ?case by simp
next
  case (Paar t1' t2')
  then show ?case
    using depth.simps(6)[of t1' t2']
      psub-trms.simps(4) if-sub-not-eq-then-psub by fastforce
next
  case (Func f t1')
  then show ?case
    using depth.simps(5)[of f t1']
      psub-trms.simps(6)[of f t1'] if-sub-not-eq-then-psub
    by fastforce
qed

```

## 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
  by (induct xs) auto

```

```

lemma atms-rev[simp]:
  shows atms (rev pi) = atms pi
  by (induct pi) auto

```

```

lemma a-not-in-atms:
  assumes a ∉ atms pi
  shows a = swapas pi a
  using assms by (induct pi) auto

```

**lemma** *swapas-pi-ineq-a*:  
**assumes** *swapas pi a ≠ a*  
**shows**  $a \in \text{atms } pi$   
**using** *a-not-in-atms assms* **by force**

**lemma** *a-ineq-swapas-pi*:  
**assumes**  $a \neq \text{swapas } pi \ a$   
**shows**  $a \in \text{atms } pi$   
**using** *a-not-in-atms assms* **by blast**

**lemma** *swapas-pi-in-atms*:  
**assumes**  $a \in \text{atms } pi$   
**shows**  $\text{swapas } pi \ a \in \text{atms } pi$   
**using** *assms* **by** (*metis a-not-in-atms swapas-rev-pi-a*)

## 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-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** *swapas pi a ≠ a*  
**shows**  $a \in ds \ [] \ pi$   
**using** *assms ds-def swapas-pi-ineq-a* **by simp**

**corollary** *ds-elem-cp*:  
**assumes**  $a \notin ds \ [] \ pi$   
**shows**  $\text{swapas } pi \ a = a$   
**using** *assms ds-elem* **by blast**

**lemma** *elem-ds*:  
**assumes**  $a \in ds \ [] \ pi$   
**shows**  $a \neq \text{swapas } pi \ a$   
**using** *assms ds-def* **by simp**

**lemma** *ds-sym*:  
**shows**  $ds \ pi1 \ pi2 = ds \ pi2 \ pi1$   
**using** *ds-def* **by auto**

**lemma** *ds-trans*:  
**assumes**  $c \in ds \ pi1 \ pi3$   
**shows**  $c \in ds \ pi1 \ pi2 \vee c \in ds \ pi2 \ pi3$   
**using** *assms ds-def a-not-in-atms swapas-pi-ineq-a* **by auto**

**lemma** *ds-cancel-pi-left*:  
**assumes**  $c \in ds \ (pi1 @ pi) \ (pi2 @ pi)$

**shows**  $swapas\ pi\ c \in ds\ pi1\ pi2$   
**using** *assms ds-def swapas-append a-ineq-swapas-pi a-not-in-atms*  
**by** (*metis (mono-tags, lifting) Un-iff mem-Collect-eq*)

**lemma** *ds-cancel-pi-right*:  
**assumes**  $swapas\ pi\ c \in ds\ pi1\ pi2$   
**shows**  $c \in ds\ (pi1@pi)\ (pi2@pi)$   
**using** *assms*  
**unfolding** *ds-def*  
**using** *a-not-in-atms swapas-append* **by** *auto*

**lemma** *ds-equality*:  
**shows**  $(ds\ []\ pi) - \{a, swapas\ pi\ a\} = (ds\ []\ ((a, swapas\ pi\ a) \# pi)) - \{swapas\ pi\ a\}$   
**using** *ds-def* **by** *auto*

**lemma** *ds-7*:  
**assumes**  $b \neq swapas\ pi\ b\ a \in ds\ []\ ((b, swapas\ pi\ b) \# pi)$   
**shows**  $a \in ds\ []\ pi$   
**using** *assms ds-def swapas-pi-in-atms a-ineq-swapas-pi swapas-rev-pi-a*  
*ds-elem elem-ds swapa.simps swapas.simps(2)*  
**by** *metis*

**lemma** *ds-cancel-pi-front*:  
**shows**  $ds\ (pi@pi1)\ (pi@pi2) = ds\ pi1\ pi2$   
**unfolding** *ds-def*  
**by** (*metis (lifting) Un-iff a-not-in-atms swapas-append swapas-inv*)

**lemma** *ds-rev-pi-pi*:  
**shows**  $ds\ ((rev\ pi1)@pi1)\ pi2 = ds\ []\ pi2$   
**unfolding** *ds-def*  
**using** *a-not-in-atms swapas-append* **by** *auto*

**lemma** *ds-rev*:  
**shows**  $ds\ []\ ((rev\ pi1)@pi2) = ds\ pi1\ pi2$   
**using** *ds-cancel-pi-front ds-rev-pi-pi* **by** *blast*

**lemma** *ds-acabb*:  
**assumes**  $a \neq b\ b \neq c\ c \neq a$   
**shows**  $ds\ [(a, b), (b, c)]\ [(a, c)] = \{a, b\}$   
**using** *assms ds-def* **by** *auto*

**lemma** *ds-baab*:  
**assumes**  $a \neq b$   
**shows**  $ds\ [(b, a)]\ [(a, b)] = \{\}$   
**using** *assms ds-def* **by** *auto*

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

**using** *assms ds-def ds-rev ds-baab* **by** *simp*

**lemma** *ds-abab*:

**shows**  $ds \ [] \ [(a, b), (a, b)] = \{\}$   
**using** *ds-def* **by** *auto*

**lemma** *ds-comm*:

**shows**  $ds \ (pi \ @ \ [(a,b)]) \ (([swapas \ pi \ a, \ swapas \ pi \ b]) \ @ \ pi) = \{\}$   
**using** *swapas-comm ds-def* **by** *simp*

**lemma** *ds-rev-pi-id*:

**shows**  $ds \ (rev \ pi \ @ \ pi) \ [] = \{\}$   
**using** *ds-rev-pi-pi[of pi <[]> elem-ds*  
*ds-sym[of <(rev pi @ pi)> <[]>]* **by** *fastforce*

**lemma** *ds-pi-rev-id*:

**shows**  $ds \ (pi \ @ \ rev \ pi) \ [] = \{\}$   
**using** *ds-rev-pi-id[of <rev pi> rev-rev-ident[of pi]*  
**by** *simp*

**lemma** *ds-swapas-eq*:

**assumes**  $ds \ pi1 \ pi2 = \{\}$   
**shows**  $swapas \ pi1 \ a = swapas \ pi2 \ a$   
**using** *assms*  
**by** (*metis ds-elem-cp ds-pi-rev-id ds-rev ds-sym emptyE swapas-append*)

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**  $set \ (flatten \ pi) = atms \ pi$   
**by** (*induct pi*) *auto*

**lemma** *ds-list-equ-ds*:

$set \ (ds-list \ pi1 \ pi2) = ds \ pi1 \ pi2$   
**using** *ds-list-def ds-def set-flatten-eq-atms* **by** *auto*

## 6 Freshness

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

**type-synonym** *fresh-envs* = (*string* × *string*) *set*

**inductive** *fresh* :: *fresh-envs* ⇒ *string* ⇒ *trm* ⇒ *bool* ( - ⊢ - # - [80,80,80] 80)

**where**

*fresh-abst-ab*[*intro!*]:  $\llbracket nabra \vdash a \# t; a \neq b \rrbracket \Longrightarrow nabra \vdash a \# Abst\ b\ t \mid$

*fresh-abst-aa*[*intro!*]:  $nabra \vdash a \# Abst\ a\ t \mid$

*fresh-unit*[*intro!*]:  $nabra \vdash a \# Unit \mid$

*fresh-atom*[*intro!*]:  $a \neq b \Longrightarrow nabra \vdash a \# Atom\ b \mid$

*fresh-susp*[*intro!*]:  $(swapas\ (rev\ pi)\ a, X) \in nabra \Longrightarrow nabra \vdash a \# Susp\ pi\ X \mid$

*fresh-paar*[*intro!*]:  $\llbracket nabra \vdash a \# t1; nabra \vdash a \# t2 \rrbracket \Longrightarrow nabra \vdash a \# Paar\ t1\ t2 \mid$

*fresh-func*[*intro!*]:  $nabra \vdash a \# t \Longrightarrow nabra \vdash a \# Func\ F\ t$

**inductive-cases** *Fresh-elim*s:

$nabra \vdash a \# Abst\ b\ t$

$nabra \vdash a \# Unit$

$nabra \vdash a \# Atom\ b$

$nabra \vdash a \# Susp\ pi\ X$

$nabra \vdash a \# Paar\ t1\ t2$

$nabra \vdash a \# Func\ F\ t$

**lemma** *fresh-swap-eqv*:

**assumes**  $nabra \vdash a \# t$

**shows**  $nabra \vdash swapas\ pi\ a \# swap\ pi\ t$

**using** *assms*

**by** (*induct arbitrary: pi rule: fresh.induct*)

(*auto simp add: swapas-append dest: swapas-rev-pi-a*)

**lemma** *fresh-swap-left*:

**assumes**  $nabra \vdash a \# swap\ pi\ t$

**shows**  $nabra \vdash swapas\ (rev\ pi)\ a \# t$

**using** *assms*

**proof**(*induct t arbitrary: a pi*)

**case** (*Abst x1 t*)

**have**  $nabra \vdash a \# swap\ pi\ (Abst\ x1\ t)$  **by** *fact*

**then have**  $(nabra \vdash a \# swap\ pi\ t \wedge a \neq swapas\ pi\ x1) \vee (a = swapas\ pi\ x1)$

**by** (*metis Fresh-elim*s(1) *swap.simp*s(4))

**moreover** {

**assume** *as*:  $nabra \vdash a \# swap\ pi\ t \wedge a \neq swapas\ pi\ x1$

**have**  $nabra \vdash swapas\ (rev\ pi)\ a \# Abst\ x1\ t$

**using** *Abst.hyps as by auto*

}

**moreover** {

**assume** *as*:  $a = swapas\ pi\ x1$

**then have**  $nabra \vdash swapas\ (rev\ pi)\ a \# Abst\ x1\ t$

```

    by (simp add: fresh-abst-aa)
  }
  ultimately show  $nabla \vdash \text{swapas} (\text{rev } \pi) a \# \text{Abst } x1 t$ 
    by argo
next
case (Susp  $x1 x2$ )
have  $nabla \vdash a \# \text{swap } \pi (\text{Susp } x1 x2)$  by fact
have  $(\text{swapas} (\text{rev } x1) (\text{swapas} (\text{rev } \pi) a), x2) \in nabla$ 
  by (metis Fresh-elim4) Susp rev-append swap.simps(3) swapas-append
then show  $nabla \vdash \text{swapas} (\text{rev } \pi) a \# \text{Susp } x1 x2$ 
  by (simp add: fresh-susp)
next
case (Atom  $x$ )
have  $nabla \vdash a \# \text{swap } \pi (\text{Atom } x)$  by fact
then have  $\text{swapas } \pi x \neq a$  by (auto elim: Fresh-elim)
then have  $x \neq \text{swapas} (\text{rev } \pi) a$  using swapas-rev- $\pi$ -b by blast
then show  $nabla \vdash \text{swapas} (\text{rev } \pi) a \# \text{Atom } x$ 
  by (simp add: fresh-atom)
qed (auto elim: Fresh-elim)

```

**lemma** *fresh-swap-right*:

```

  assumes  $nabla \vdash \text{swapas} (\text{rev } \pi) a \# t$ 
  shows  $nabla \vdash a \# \text{swap } \pi t$ 
  using assms fresh-swap-eqt[of  $nabla \langle \text{swapas} (\text{rev } \pi) a \rangle t \pi$ ]
    swapas-rev-swapas-id[of  $\pi a$ ] by simp

```

**lemma** *fresh-weak*:

```

  assumes  $nabla1 \vdash a \# t$ 
  shows  $(nabla1 \cup nabla2) \vdash a \# t$ 
  using assms
  by (induct rule: fresh.induct)(auto)

```

**lemma** *ds-empty-fresh-1*:

```

  assumes  $ds \ \pi1 \ \pi2 = \{\}$ 
  shows  $nabla \vdash \text{swapas } \pi1 a \# t \implies nabla \vdash \text{swapas } \pi2 a \# t$ 
  using assms ds-swapas-eq by simp

```

**lemma** *ds-empty-fresh-2*:

```

  assumes  $ds \ \pi1 \ \pi2 = \{\}$ 
  shows  $nabla \vdash a \# \text{swap } \pi1 t \implies nabla \vdash a \# \text{swap } \pi2 t$ 
  using assms

```

**proof** –

```

  assume assm1:  $nabla \vdash a \# \text{swap } \pi1 t$ 
  have i:  $\text{swapas} (\text{rev } \pi1) a = \text{swapas} (\text{rev } \pi2) a$ 
    using assms ds-swapas-eq swapas-rev- $\pi$ -a by metis
  with assms have ii:  $nabla \vdash \text{swapas} (\text{rev } \pi2) a \# t$ 
    using fresh-swap-left[OF assm1] i by simp
  show ?thesis
    using fresh-swap-right[OF ii] by simp

```

qed

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

**inductive-cases** *Equ-elim*s:

*nabla*  $\vdash Atom\ a \approx Atom\ b$   
*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*  
**using** *assms* **by** (*rule equ.induct, simp-all*)

**lemma** *rev-pi-pi-equ*:

**shows**  $(nabla \vdash swap\ (rev\ pi)\ (swap\ pi\ t) \approx t)$   
**proof**(*induct* *t*)  
**case** (*Susp* *pi'* *X*)  
**then show** *?case*  
**using** *ds-cancel-pi-left*[*of* -  $\langle rev\ pi\ @\ pi \rangle$  -  $\langle [] \rangle$ ]  
*ds-rev-pi-pi elem-ds ds-rev-pi-id* **by** *auto*  
**qed** (*auto*)

**lemma** *equ-pi-right*:

**assumes**  $\forall a \in ds\ []\ pi. nabla \vdash a \# t$   
**shows** *nabla*  $\vdash t \approx swap\ pi\ t$   
**using** *assms*  
**proof**(*induct* *t* *arbitrary: pi*)

```

case (Abst b t')
have swapas pi b = b  $\vee$  swapas pi b  $\neq$  b by simp
moreover
{ assume eq: swapas pi b = b
  have nabla b  $\vdash$  Abst b t'  $\approx$  Abst b (swap pi t')
    by (metis Abst.hyps Abst.premis Fresh-elimis(1) elem-ds eq equ-abst-aa)
  then have nabla b  $\vdash$  Abst b t'  $\approx$  swap pi (Abst b t') using eq by simp
}
moreover
{ assume neg: b  $\neq$  swapas pi b
  obtain c where c-def: c = swapas pi b by simp
  have c  $\in$  ds  $\square$  pi
    by (metis append.right-neutral c-def ds-cancel-pi-front ds-cancel-pi-left ds-elem
neg)
  then have one: nabla b  $\vdash$  c # Abst b t' using assms Abst.premis by auto
  have two: b  $\neq$  c using neg c-def by blast
  have nabla b  $\vdash$  c # t' using one two by cases auto
  have nabla b  $\vdash$  Abst b t'  $\approx$  Abst c (swap pi t')
  proof(rule equ-abst-ab)
    show b  $\neq$  c using neg c-def by blast
    have b-is-swap: b = swapas (rev pi) c using c-def by simp
    show nabla b  $\vdash$  b # swap pi t'
    using Abst.premis Fresh-elimis(1) ds-elem fresh-swap-right neg swapas-rev-pi-a
by metis
  show nabla b  $\vdash$  t'  $\approx$  swap [(b, c)] (swap pi t')
  proof –
    have fresh-ext:
       $\forall a \in ds \square ((b,c) \# pi). nabla b$   $\vdash$  a # t'
    proof (rule ballI)
      fix a assume A: a  $\in$  ds  $\square$   $((b,c) \# pi)$ 
      then have a = c  $\vee$  a  $\in$  ds  $\square$  pi
        using c-def ds-7 neg by auto
      then show nabla b  $\vdash$  a # t'
      proof
        assume a = c
        with  $\langle nabla b \vdash c \# t' \rangle$  show ?thesis by simp
      next
        assume a  $\in$  ds  $\square$  pi
        show ?thesis
        proof(rule Fresh-elimis(1))[of nabla a b t'], goal-cases)
          case 1
            then show ?case
              using Abst.premis  $\langle a \in ds \square pi \rangle$ 
            by simp
          next
            case 2
              then show ?case by simp
          next
            case 3

```

```

      have  $a \neq \text{swapas } ((b, c) \# \text{pi}) a$ 
        using  $A \text{ elem-ds}$ 
        by  $\text{blast}$ 
      hence  $a \neq b$  using  $c\text{-def } A$ 
        by  $(\text{auto simp add: if-split})$ 
      hence  $\text{False}$  using  $\exists$  by  $\text{simp}$ 
      then show  $?case$  by  $\text{simp}$ 
    qed
  qed
  qed
  from  $\text{Abst.hyps}[OF \text{fresh-ext}]$ 
  have  $\text{nabla} \vdash t' \approx \text{swap } (((b,c))@pi) t'$ 
    by  $\text{simp}$ 
  moreover have  $\text{swap } (((b,c))@pi) t' = \text{swap } [(b,c)] (\text{swap } pi t')$ 
    using  $\text{swap-append}$  by  $\text{blast}$ 
  ultimately show  $?thesis$  by  $\text{simp}$ 
  qed
  qed
  then have  $\text{nabla} \vdash \text{Abst } b t' \approx \text{swap } pi (\text{Abst } b t')$  using  $\text{neq } c\text{-def}$  by  $\text{force}$ 
}
ultimately show  $?case$  by  $\text{argo}$ 
next
case  $(\text{Susp } pi' X)$ 
have  $\forall a \in ds \ []. pi. \text{nabla} \vdash a \# \text{Susp } pi' X$  by  $\text{fact}$ 
then have  $\text{nabla} \vdash \text{Susp } pi' X \approx \text{Susp } (pi @ pi') X$ 
  using  $\text{Fresh-elim}(4) \text{ equ-susp } ds\text{-def } ds\text{-cancel-pi-left}$ 
  by  $(\text{metis } (\text{lifting}) \text{ append-self-conv2 } \text{swapas-rev-pi-a})$ 
then show  $?case$  by  $\text{simp}$ 
next
case  $\text{Unit}$ 
then show  $?case$ 
  using  $\text{equ-unit } \text{swap.simps}(2)$  by  $\text{force}$ 
next
case  $(\text{Atom } b)$ 
then show  $?case$ 
  apply  $\text{simp}$ 
  apply  $(\text{rule } \text{equ-atom})$ 
  using  $\text{Fresh-elim}(3) \text{ ds-elem-cp}$ 
  by  $\text{metis}$ 
next
case  $(\text{Paar } t1 t2)$ 
then have  $\text{hypsp1}: (\forall a \in ds \ []. pi. \text{nabla} \vdash a \# t1)$ 
  and  $\text{hypsp2}: (\forall a \in ds \ []. pi. \text{nabla} \vdash a \# t2)$ 
  using  $\text{Paar.prem } \text{Fresh-elim}(5)$  by  $\text{meson+}$ 
have  $\text{nabla} \vdash \text{Paar } t1 t2 \approx \text{Paar } (\text{swap } pi t1) (\text{swap } pi t2)$ 
  using  $\text{Paar.hyps}(1,2) \text{ hypsp1 } \text{hypsp2}$  by  $\text{blast}$ 
then show  $?case$  by  $\text{simp}$ 
next
case  $(\text{Func } f t')$ 

```

```

then have hypsf:  $\forall a \in ds \ [] \ pi. \ nabla \vdash \ a \ \# \ t'$  using fresh-func
  by (metis Fresh-elim(6))
have  $nabla \vdash \ Func \ f \ t' \approx \ Func \ f \ (swap \ pi \ t')$ 
  using Func.hyps hypsf by blast
then show ?case
  by simp
qed

```

lemma pi-comm:

```

shows  $nabla \vdash \ (swap \ (pi \ @ \ [(a,b)]) \ t) \approx \ (swap \ ([ (swapas \ pi \ a, \ swapas \ pi \ b)] \ @ \ pi) \ t)$ 
proof(induct t)
  case (Abst c t)
  then show ?case using swapas-comm by force
next
  case (Susp  $\pi' \ X$ )
  have rew1:
     $swap \ (pi \ @ \ [(a,b)]) \ (Susp \ \pi' \ X) = Susp \ (pi \ @ \ [(a,b)] \ @ \ \pi') \ X$ 
  by simp
  have rew2:
     $swap \ ([ (swapas \ pi \ a, \ swapas \ pi \ b)] \ @ \ pi) \ (Susp \ \pi' \ X)$ 
     $= Susp \ ([ (swapas \ pi \ a, \ swapas \ pi \ b)] \ @ \ pi \ @ \ \pi') \ X$ 
  by simp
  have fresh:
     $\forall c \in ds \ (pi \ @ \ [(a,b)] \ @ \ \pi') \ ([ (swapas \ pi \ a, \ swapas \ pi \ b)] \ @ \ pi \ @ \ \pi'). \ (c, \ X) \in$ 
     $nabla$ 
  using swapas-append swapas-comm ds-def by simp
  from rew1 rew2 fresh
  show ?case
    using equ-susp by simp
next
  case Unit
  then show ?case by force
next
  case (Atom c)
  then show ?case
    using equ-atom swapas-comm by auto
next
  case (Paar t1 t2)
  then show ?case by force
next
  case (Func f t)
  then show ?case by force
qed

```

lemma l3-jud:

```

assumes  $(nabla \vdash \ t1 \approx \ t2)$ 
shows  $(nabla \vdash \ a \ \# \ t1) \longrightarrow (nabla \vdash \ a \ \# \ t2)$ 
using assms

```

```

proof(induction rule: equ.induct)
  case (equ-abst-ab b c nabla t2 t1)
  then show ?case
    using fresh-swap-left Fresh-elim(1) fresh-abst-aa fresh-abst-ab rev-singleton-conv
    swapa.simps swapas.simps(1,2)
    by metis
  next
  case (equ-abst-aa nabla t1 t2 b)
  then show ?case
    using fresh-swap-left Fresh-elim(1) fresh-abst-aa fresh-abst-ab
    by metis
  next
  case (equ-unit nabla)
  then show ?case
    by blast
  next
  case (equ-atom b c nabla)
  then show ?case
    by simp
  next
  case (equ-susp pi1 pi2 X nabla)
  then show ?case
    using Fresh-elim(4) ds-elim-cp ds-rev fresh-susp swapas-append swapas-rev-pi-a
    by metis
  next
  case (equ-paar nabla t1 t2 s1 s2)
  then show ?case
    using Fresh-elim(5) by blast
  next
  case (equ-func nabla t1 t2 f)
  then show ?case
    using Fresh-elim(6) by blast
qed

```

```

lemma ds-empty-equiv-1:
  assumes ds pi1 pi2 = {}
  shows nabla ⊢ swap pi1 t1 ≈ t2 ⇒ nabla ⊢ swap pi2 t1 ≈ t2
  using assms
proof(induction t1 arbitrary: t2)
  case (Abst a t1')
  then obtain b t2'
    where t2: t2 = Abst b t2'
    by(auto elim: equ.cases)
  with Abst have i: nabla ⊢ Abst (swapas pi1 a) (swap pi1 t1') ≈ Abst b t2'
    using swapa.simps(4) by simp
  then show ?case
proof(cases ⟨swapas pi1 a = b⟩)
  case True
    then have nabla ⊢ swap pi1 t1' ≈ t2'

```

```

    using Abst i Equ-elim3(6) by blast
  then have ii: nabl a ⊢ swap pi2 t1' ≈ t2'
    using Abst by simp
  then have nabl a ⊢ Abst (swapas pi2 a) (swap pi2 t1') ≈ Abst b t2'
    using True ds-swapas-eq[OF Abst(3), of a] equ-abst-aa[OF ii, of ⟨swapas pi2
a⟩]
    by simp
  then show ?thesis
    using t2 by simp
next
case False
then have equ-ab-pi1: nabl a ⊢ swap pi1 t1' ≈ swap [(swapas pi1 a, b)] t2'
  and fresh-ab-pi1: nabl a ⊢ swapas pi1 a ‡ t2'
  using i Equ-elim3(7)[of nabl a ⟨swapas pi1 a⟩ ⟨swap pi1 t1'⟩ b t2']
  by blast+
have equ-ab-pi2: nabl a ⊢ swap pi2 t1' ≈ swap [(swapas pi2 a, b)] t2'
  using equ-ab-pi1 ds-swapas-eq[OF Abst(3), of a]
  Abst(3) Abst(1)[of ⟨swap [(swapas pi1 a, b)] t2'⟩] by auto
have fresh-ab-pi2: nabl a ⊢ swapas pi2 a ‡ t2'
  using fresh-ab-pi1 ds-swapas-eq[OF Abst(3), of a] by simp
with equ-ab-pi2 have nabl a ⊢ Abst (swapas pi2 a) (swap pi2 t1') ≈ Abst b t2'
  using equ-abst-ab[OF False, of nabl a t2'] ds-swapas-eq[OF Abst(3), of a] by
simp
  then show ?thesis using t2 by simp
qed
next
case (Susp pi X)
then obtain pi'
  where t2: t2 = Susp pi' X
  by (auto elim: equ.cases)
with Susp have i: nabl a ⊢ Susp (pi1 @ pi) X ≈ Susp pi' X
  by simp
then have ∀ c ∈ ds (pi1 @ pi) pi'. (c, X) ∈ nabl a
  using Equ-elim3(3)[OF i] by auto
with Susp have ∀ c ∈ ds (pi2 @ pi) pi'. (c, X) ∈ nabl a
  using ds-cancel-pi-left ds-sym ds-trans
  by blast
then have nabl a ⊢ Susp (pi2 @ pi) X ≈ Susp pi' X
  using equ-susp by simp
then show ?case using t2 swap.simps(3) by simp
next
case (Atom a)
then show ?case
  using ds-swapas-eq[OF Atom(2), of a]
  by (auto elim: equ.cases)
qed (auto elim: equ.cases)

lemma ds-empty-equiv-2:
  assumes ds pi1 pi2 = {}

```

**shows**  $nabla \vdash t1 \approx \text{swap } \pi1 \ t2 \implies \nabla \vdash t1 \approx \text{swap } \pi2 \ t2$   
**using** *assms*  
**proof**(*induction t2 arbitrary: pi1 pi2 t1*)  
  **case** (*Abst b t2'*)  
  **then obtain** *a t1'*  
    **where** *t1: t1 = Abst a t1'*  
    **by**(*auto elim: equ.cases*)  
  **with** *Abst have i: nabla \vdash Abst a t1' \approx Abst (swapas pi1 b) (swap pi1 t2')*  
    **using** *swap.simps(4)* **by** *simp*  
  **then show** *?case*  
  **proof**(*cases \langle swapas pi1 b = a \rangle*)  
    **case** *True*  
    **then have**  $nabla \vdash t1' \approx \text{swap } \pi1 \ t2'$   
      **using** *Abst i Equ-elim(6)* **by** *blast*  
    **then have** *ii: nabla \vdash t1' \approx \text{swap } \pi2 \ t2'*  
      **using** *Abst* **by** *simp*  
    **then have**  $nabla \vdash \text{Abst } a \ t1' \approx \text{Abst } (\text{swapas } \pi2 \ b) \ (\text{swap } \pi2 \ t2')$   
      **using** *True ds-swapas-eq[OF Abst(3), of b] equ-abst-aa[OF ii, of \langle swapas pi2*  
      *b \rangle]*  
      **by** *simp*  
    **then show** *?thesis*  
      **using** *t1* **by** *simp*  
  **next**  
  **case** *False*  
  **then have** *equ-ab: nabla \vdash t1' \approx \text{swap } [(a, swapas pi1 b)] (\text{swap } \pi1 \ t2')*  
    **and** *fresh-ab-pi1: nabla \vdash a \# \text{swap } \pi1 \ t2'*  
    **using** *i Equ-elim(7)* **by** *blast+*  
  
  **then have** *equ-ab-pi1: nabla \vdash t1' \approx \text{swap } [(a, swapas pi1 b)] @ \pi1 \ t2'*  
    **using** *swap-append[of \langle [(a, swapas pi1 b)] \rangle \pi1 \ t2']* **by** *simp*  
  
  **have** *ds-empty: ds [(a, swapas pi1 b)] @ \pi1 [(a, swapas pi2 b)] @ \pi2 = \{\}*  
    **using** *Abst(3) ds-swapas-eq[OF Abst(3), of b]*  
    *ds-cancel-pi-front[of \langle [(a, swapas pi1 b)] \rangle \pi1 \ \pi2]* **by** *simp*  
  
  **hence**  $nabla \vdash t1' \approx \text{swap } [(a, \text{swapas } \pi2 \ b)] @ \pi2 \ t2'$   
    **using** *Abst(1)[OF equ-ab-pi1 ds-empty]* **by** *simp*  
  **hence** *equ-ab-pi2: nabla \vdash t1' \approx \text{swap } [(a, swapas pi2 b)] (\text{swap } \pi2 \ t2')*  
    **using** *swap-append[of \langle [(a, swapas pi2 b)] \rangle \pi2 \ t2']* **by** *simp*  
  
  **have** *fresh-ab-pi2: nabla \vdash a \# \text{swap } \pi2 \ t2'*  
    **using** *ds-empty-fresh-2 Abst(3) fresh-ab-pi1* **by** *auto*  
  **with** *equ-ab-pi2* **have**  $nabla \vdash \text{Abst } a \ t1' \approx \text{Abst } (\text{swapas } \pi2 \ b) \ (\text{swap } \pi2 \ t2')$   
    **using** *equ-abst-ab ds-swapas-eq False assms Abst(3)* **by** *auto*  
  **then show** *?thesis* **using** *t1* **by** *simp*  
  **qed**  
**next**  
  **case** (*Susp pi' X*)  
  **then obtain** *pi*

```

    where t1: t1 = Susp pi X
    by (auto elim: equ.cases)
  with Susp have i: nbla ⊢ Susp pi X ≈ Susp (pi1 @ pi') X
    by simp
  then have ∀ c ∈ ds pi (pi1 @ pi'). (c, X) ∈ nbla
    using Equ-elim3[OF i] by auto
  with Susp have ∀ c ∈ ds pi (pi2 @ pi'). (c, X) ∈ nbla
    using ds-cancel-pi-left ds-sym ds-trans
    by blast
  then have nbla ⊢ Susp pi X ≈ Susp (pi2 @ pi') X
    using equ-susp by simp
  then show ?case using t1 swap.simps(3) by simp
next
  case (Atom a)
  then show ?case
    using ds-swaps-eq by (auto elim: equ.cases)
qed (auto elim: equ.cases)

lemma equ-equivariance:
  assumes nbla ⊢ t1 ≈ t2
  shows nbla ⊢ swap pi t1 ≈ swap pi t2
  using assms
proof(induct rule: equ.induct)
  case (equ-abst-ab a b nbla t2' t1')
  have i: nbla ⊢ swaps pi a ‡ swap pi t2'
    using fresh-swap-eqv[OF equ-abst-ab(2), of pi] by simp

  have nbla ⊢ swap pi t1' ≈ swap (pi @ [(a,b)]) t2'
    using equ-abst-ab(4) swap-append[of pi ⟨[(a,b)]⟩ t2'] by simp
  then have nbla ⊢ swap pi t1' ≈ swap [(swaps pi a, swaps pi b)] @ pi t2'
    using ds-empty-equiv-2[OF ds-comm[of pi a b]] by auto
  then have ii: nbla ⊢ swap pi t1' ≈ swap [(swaps pi a, swaps pi b)] (swap pi
t2')
    using ds-empty-equiv-2[OF ds-comm[of pi a b]]
    swap-append[of ⟨[(swaps pi a, swaps pi b)]⟩ pi t2'] by simp

  have iii: swaps pi a ≠ swaps pi b
    using equ-abst-ab.hyps(1) swaps-rev-pi-a by blast

  with i ii
  have nbla ⊢ Abst (swaps pi a) (swap pi t1') ≈ Abst (swaps pi b) (swap pi t2')
    using equ.equ-abst-ab by auto
  hence nbla ⊢ swap pi (Abst a t1') ≈ swap pi (Abst b t2')
    using swap.simps(4) by simp
  then show ?case by simp
next
  case (equ-abst-aa nbla t1' t2' a)
  then show ?case
    using equ.equ-abst-aa by simp

```

```

next
  case (equ-susp pi1 pi2 X nabra)
  then show ?case using ds-cancel-pi-front
    by auto
qed (auto)

lemma swap-inv-side:
  shows nabra ⊢ swap pi t1 ≈ t2 = nabra ⊢ t1 ≈ swap (rev pi) t2
proof

  {assume assm1: nabra ⊢ swap pi t1 ≈ t2
  from equ-equivariance[OF assm1, of ⟨rev pi⟩]
  have nabra ⊢ swap (rev pi @ pi) t1 ≈ swap (rev pi) t2
    using swap-append[of ⟨rev pi⟩ pi t1] by simp
  then show nabra ⊢ t1 ≈ swap (rev pi) t2
    using ds-empty-equiv-1[OF ds-rev-pi-id[of pi], of nabra t1 ⟨swap (rev pi) t2⟩]
    swap-id[of t1] by simp}

  {assume assm2: nabra ⊢ t1 ≈ swap (rev pi) t2
  from equ-equivariance[OF assm2, of pi]
  have nabra ⊢ swap pi t1 ≈ swap (pi @ rev pi) t2
    using swap-append[of pi ⟨rev pi⟩ t2] by simp
  then show nabra ⊢ swap pi t1 ≈ t2
    using ds-empty-equiv-2[OF ds-pi-rev-id[of pi], of nabra ⟨swap pi t1⟩ t2]
    swap-id[of t2] by simp}
qed

lemma equ-swap-abba:
  assumes n = depth t1
  shows nabra ⊢ swap [(a,b)] t1 ≈ t2 ⟹ nabra ⊢ t1 ≈ swap [(b,a)] t2
proof -
  assume nabra ⊢ swap [(a,b)] t1 ≈ t2
  with ds-baab[of b a]
  have nabra ⊢ swap [(b,a)] t1 ≈ t2
    using ds-empty-equiv-1 by blast
  then show ?thesis
    using swap-inv-side[of nabra ⟨[(b,a)]⟩] by simp
qed

lemma equ-equiv-pi:
  assumes ∀ a ∈ ds pi1 pi2. nabra ⊢ a ‡ t
  shows nabra ⊢ swap pi1 t ≈ swap pi2 t
  using assms equ-pi-right[of ⟨rev pi1 @ pi2⟩ nabra t]
    swap-inv-side[of nabra pi1 t ⟨swap pi2 t⟩] ds-rev swap-append by simp

lemma equ-symm:
  shows (nabra ⊢ t1 ≈ t2) ⟹ (nabra ⊢ t2 ≈ t1)
proof(induction rule: equ.induct)
  case (equ-abst-ab a b nabra t2' t1')

```

```

have i:  $nabla \vdash b \# \text{swap} [(a, b)] t2'$ 
  using fresh-swap-eqvt[of  $nabla a t2' [(a, b)]$ ] equ-abst-ab(2) by auto
with equ-abst-ab(4)
have b-fresh:  $nabla \vdash b \# t1'$ 
  using l3-jud by blast
from equ-abst-ab(4)
have ii:  $nabla \vdash t2' \approx \text{swap} [(b, a)] t1'$ 
  using equ-swap-abba by simp
show ?case
  using equ.equ-abst-ab[OF equ-abst-ab(1)[symmetric] b-fresh ii] by simp
next
  case (equ-abst-aa  $nabla t1' t2' a$ )
  then show ?case
    using equ.equ-abst-aa[OF equ-abst-aa(2), of a] by simp
next
  case (equ-unit  $nabla$ )
  then show ?case by auto
next
  case (equ-atom  $a b nabla$ )
  then show ?case by auto
next
  case (equ-susp  $\pi1 \pi2 X nabla$ )
  then show ?case
    using ds-sym by auto
next
  case (equ-paar  $nabla t1' t2' s1' s2'$ )
  then show ?case by auto
next
  case (equ-func  $nabla t1' t2' f$ )
  then show ?case
    using equ.equ-func[OF equ-func(2), of f] by simp
qed

lemma equ-trans:
  assumes  $nabla \vdash t1 \approx t2$   $nabla \vdash t2 \approx t3$ 
  shows  $nabla \vdash t1 \approx t3$ 
  using assms
proof(induction  $\langle \text{depth } t1 \rangle$  arbitrary:  $t1 t2 t3$  rule: nat-less-induct)
  case 1
  note IH = this
  have IH-usable:
     $\text{depth } t1' < \text{depth } t1 \implies nabla \vdash t1' \approx t2' \implies nabla \vdash t2' \approx t3' \implies nabla$ 
     $\vdash t1' \approx t3'$ 
    for  $t1' t2' t3'$  using IH by blast
  have  $nabla \vdash t1 \approx t2 \implies nabla \vdash t2 \approx t3 \implies nabla \vdash t1 \approx t3$ 
  proof–
    assume t12:  $nabla \vdash t1 \approx t2$  and t23:  $nabla \vdash t2 \approx t3$ 
    show ?thesis
    proof(cases rule: equ.cases[OF  $\langle nabla \vdash t1 \approx t2 \rangle$ )

```

```

case (1 a b nabla t2' t1')
from 1(2) have deptht1: depth t1' < depth t1
  using depth.simps(4) by auto
from t23 show ?thesis
proof(cases rule: equ.cases)
  case (equ-abst-ab b c t3' t2')
  then show ?thesis
  proof(cases a = c)
    case True
    have i: nabla ⊢ swap [(c,b)] t2' ≈ swap ([[c,b]]@[b,c]) t3'
      using equ-equivariance[OF equ-abst-ab(5), of ⟨[(c,b)]⟩ 1(1)
        swap-append[of ⟨[(c,b)]⟩ ⟨[(b,c)]⟩ t3'] by simp
    have ii: nabla ⊢ swap [(c,b)] t2' ≈ t3'
      using ds-empty-equiv-2[OF ds-baab-id[OF equ-abst-ab(3)] i] by simp
    with equ-abst-ab 1 True
    have iii: nabla ⊢ t1' ≈ swap [(c, b)] t2' by blast
    with ii have nabla ⊢ t1' ≈ t3'
      using IH-usable[OF deptht1, of ⟨swap [(c, b)] t2'⟩ t3'] 1(1)
      by simp
    then show ?thesis using True equ-abst-ab(2) 1(1,2) by auto
  next
  case False

  have deptht2: depth (swap [(a,b)] t2') < depth t1
    using depth.simps(4) equ-depth[OF 1(6)] swap-depth 1(3) deptht1
    equ-abst-ab(1) by simp

  from equ-abst-ab(1,4,5) 1(1,3,5)
    have ii: nabla ⊢ a ‡ swap [(b,c)] t3'
      using l3-jud by simp

  then have a-fresh-t3: nabla ⊢ a ‡ t3'
    using fresh-swap-left[OF ii] False equ-abst-ab 1 by simp
  have b-fresh-t3: nabla ⊢ b ‡ t3'
    using equ-abst-ab(4) 1(1) by simp

  from 1(1) equ-abst-ab(5)
  have nabla ⊢ swap [(a,b)] t2' ≈ swap ([[a,b]]@[b,c]) t3'
    using equ-equivariance[OF equ-abst-ab(5)]
    swap-append[of ⟨[(a, b)]⟩ ⟨[(b,c)]⟩ t3'] by simp
  then have iii: nabla ⊢ swap [(a,b)] t2' ≈ swap [(a,b),(b,c)] t3'
    by simp

  have ds [(a,b), (b,c)] [(a,c)] = {a,b}
    using ds-acabc equ-abst-ab 1 False by simp
  with a-fresh-t3 b-fresh-t3
  have iv: nabla ⊢ swap [(a,b),(b,c)] t3' ≈ swap [(a,c)] t3'
    using equ-equiv-pi by simp

```

$t3'$ ]

```

with 1(1) iii have  $nabla \vdash \text{swap} [(a,b)] t2' \approx \text{swap} [(a,c)] t3'$ 
using IH-usable[OF deptht2, of  $\langle \text{swap} [(a,b),(b,c)] \rangle t3' \langle \text{swap} [(a,c)]$ 
by simp

with 1(1,3,6) have  $nabla \vdash t1' \approx \text{swap} [(a,c)] t3'$ 
using IH-usable[OF deptht1, of  $\langle \text{swap} [(a,b)] t2' \rangle \langle \text{swap} [(a,c)] t3' \rangle$ 
equ-abst-ab(1) by simp

with 1(1,2) equ-abst-ab(2)
show ?thesis using equ.equ-abst-ab[OF False a-fresh-t3] by simp
qed
next
case (equ-abst-aa  $t2' t3' b$ )
from this(3) 1(1)
have i:  $nabla \vdash \text{swap} [(a,b)] t2' \approx \text{swap} [(a,b)] t3'$ 
using equ-equivariance[of  $nabla t2' t3' \langle [(a,b)] \rangle$ ] by simp

have  $nabla \vdash b \# \text{swap} [(a,b)] t2'$ 
using 1(1,3,4,5) equ-abst-aa(1) fresh-swap-eqv[of  $nabla a t2' \langle [(a,b)] \rangle$ ]
by auto
then have ii:  $nabla \vdash b \# \text{swap} [(a,b)] t3'$ 
using l3-jud[OF i, of b] by simp
have  $nabla \vdash \text{swapas} [(a,b)] b \# t3'$ 
using fresh-swap-left[OF ii] by simp
then have a-fresh-t3:  $nabla \vdash a \# t3'$ 
using 1(3,4) equ-abst-aa(1) by simp

have is-equ:  $nabla \vdash t1' \approx \text{swap} [(a,b)] t3'$ 
using 1 i IH-usable[OF deptht1, of  $\langle \text{swap} [(a,b)] t2' \rangle \langle \text{swap} [(a,b)] t3' \rangle$ 
equ-abst-aa(1)] by auto

from a-fresh-t3 is-equ show ?thesis
using 1 equ-abst-aa(1,2) equ.equ-abst-ab by simp
qed (auto simp add: 1 t12)
next
case (2  $nabla t1' t2' a$ )
have deptht1:  $\text{depth } t1' < \text{depth } t1$ 
using depth.simps(4) 2(2) by simp
from t23 show ?thesis
proof(cases rule: equ.cases)
case (equ-abst-ab  $a c t3' t2'$ )
have  $nabla \vdash t1' \approx \text{swap} [(a,c)] t3'$ 
using 2(1,2,3,4) IH-usable[OF deptht1, of  $\langle t2' \rangle \langle \text{swap} [(a,c)] t3' \rangle$ ]
equ-abst-ab(1,4,5) by simp
then show ?thesis
using equ.equ-abst-ab[OF equ-abst-ab(3)]
2(1,2,3) equ-abst-ab(1,2,4) by auto
next

```

```

      case (equ-abst-aa t2' t3' a)
      then show ?thesis
        using 1 2(1,2,3,4) by auto
      qed (auto simp add: 2)
next
case (3 nabla)
from t23 show ?thesis
proof(cases rule: equ.cases)
  case equ-unit
  then show ?thesis using t12 by auto
qed (auto simp add: 3)
next
case (4 a b nabla)
from t23 show ?thesis
proof(cases rule: equ.cases)
  case (equ-atom a b)
  then show ?thesis
    using t12 by auto
qed (auto simp add: 4)
next
case (5 pi1 pi2 X nabla)
from t23 show ?thesis
proof(cases rule: equ.cases)
  case (equ-susp pi1 pi2 X)
  then show ?thesis
    using ds-trans 5 by blast
qed (auto simp add: 5)
next
case (6 nabla t1' t2' s1' s2')
from t23 show ?thesis
proof(cases rule: equ.cases)
  case (equ-paar t2' t3' s2' s3')
  have deptht1: depth t1' < depth t1
    using 6(2) by simp
  have depths1: depth s1' < depth t1
    using 6(2) by simp
  have a: nabla ⊢ t1' ≈ t2'
    using 6(3,4) equ-paar by auto
  have b: nabla ⊢ s1' ≈ s2'
    using 6 equ-paar by auto
  from a have t1-equal-t3: nabla ⊢ t1' ≈ t3'
    using IH-usable[of t1' t2' t3] equ-paar(3) 6(1) deptht1
    by simp
  from b have s1-equal-s3: nabla ⊢ s1' ≈ s3'
    using IH-usable[of s1' s2' s3] equ-paar(4) 6(1) depths1
    by simp
  from t1-equal-t3 s1-equal-s3 show ?thesis
    using equ.equ-paar[of nabla t1' t3' s1' s3] 6(1,2) equ-paar(2)
    by simp

```

```

    qed (auto simp add: 6)
  next
  case (7 nabla t1 t2 F)
  from t23 show ?thesis
  proof(cases rule: equ.cases)
    case (equ-func t1 t2 F)
    then show ?thesis
      using 1 7(1,2,3,4) by auto
    qed (auto simp add: 7)
  qed
  qed
  then show ?case
    using IH(2,3) by blast
  qed

```

```

lemma pi-right-equ-help:
  assumes (n = depth t)
  shows nabla  $\vdash$   $t \approx \text{swap } \pi \ t \implies \forall a \in ds \ [] \ \pi. \ nabla \vdash a \# t$ 
  using assms
  proof(induction n arbitrary: t pi rule: nat-less-induct)
    case (1 n)
    note IH = this
    then show ?case
      proof(cases rule: equ.cases[OF  $\langle nabla \vdash t \approx \text{swap } \pi \ t \rangle$ ])
        case (1 b c nabla t2 t1)
        have deptht1: depth t1 < n
          using 1 depth.simps(4) IH by simp
        have swap-pi-t1-is-t2: swap pi t1 = t2
          using 1(2,3) by auto
        have swap-pi-b-is-c: swapas pi b = c
          using 1(2,3) by auto
        with swap-pi-t1-is-t2 have nabla  $\vdash$  t1  $\approx$  swap [(b, swapas pi b)] (swap pi
t1)
          using 1(1,6) by simp
        then have nabla  $\vdash$  t1  $\approx$  swap [(b, swapas pi b)] @ pi t1
          using swap-append[of  $\langle [(b, swapas pi b)] \rangle \ \pi \ t1$ ] by simp
        with deptht1 have  $\forall a \in ds \ [] \ ((b, swapas pi b) \# \pi). \ nabla \vdash a \# t1$ 
          using 1.IH 1 by auto
        then have ds-minus-bc:  $\forall a \in ds \ [] \ \pi - \{b, c\}. \ nabla \vdash a \# t1$ 
          using ds-equality swap-pi-b-is-c by auto
        have c-fresh-t1: nabla  $\vdash$  c  $\#$  t1
          using 1 equ-symm l3-jud fresh-swap-eqvt Equ-elim equ-abst-ab by meson
        with ds-minus-bc have  $\forall a \in ds \ [] \ \pi - \{b\}. \ nabla \vdash a \# t1$ 
          by auto
        then have ds-minus-b:  $\forall a \in ds \ [] \ \pi - \{b\}. \ nabla \vdash a \# \text{Abst } b \ t1$ 
          using fresh-abst-ab by blast
        have b-fresh-abst-t1: nabla  $\vdash$  b  $\#$  Abst b t1
          using fresh-abst-aa by simp
        have  $\forall a \in ds \ [] \ \pi. \ nabla \vdash a \# \text{Abst } b \ t1$ 

```

```

proof(rule, goal-cases)
  case (1 a)
  then show ?case
    using b-fresh-abst-t1 ds-minus-b
    by (cases a=b, auto)
  qed
  with 1 show ?thesis
    by auto
next
  case (2 nabla t1 t2 b)
  have deptht1: depth t1 < n
    using 2 depth.simps(4) IH by simp
  have swap-pi-t1-is-t2: swap pi t1 = t2
    using 2(2,3) by auto
  from swap-pi-t1-is-t2 deptht1
  have  $\forall a \in ds \ [] pi. nabla \vdash a \# t1$ 
    using 1.IH 2(1,4) by auto
  then show ?thesis
    using 2(1,2,3) elem-ds by fastforce
next
  case (3 nabla)
  then show ?thesis
    by blast
next
  case (4 a b nabla)
  then show ?thesis
    using elem-ds by force
next
  case (5 pi1 pi2 X nabla)
  have swap pi t = Susp (pi@pi1) X
    using 5(2) by auto
  also have ... = Susp pi2 X
    using 5(3) calculation by simp
  then have pi@pi1 = pi2 by simp
  hence  $\forall c \in ds \ pi1 (pi@pi1). (c, X) \in nabla$ 
    using 5(4) by simp
  then show ?thesis
    using 5(1,2) fresh-susp ds-cancel-pi-right[of - - [] -]
    by simp
next
  case (6 nabla t1 t2 s1 s2)
  have deptht1: depth t1 < n
    using 6 depth.simps(5) IH
    by simp
  have depths1: depth s1 < n
    using 6 depth.simps(5) IH
    by simp
  from deptht1 depths1 show ?thesis
    using 1.IH 6 by auto

```

```

next
  case ( $\gamma$  nabra t1 t2 f)
  have deptht1: depth t1 < n
    using  $\gamma(2)$  depth.simps(5) IH
    by auto
  have nabra  $\vdash$  t1  $\approx$  swap pi t1
    using  $\gamma$  by simp
  then have  $\forall a \in ds \ [] pi. nabra \vdash a \# t1$ 
    using  $\gamma(1)$  IH deptht1 by simp
  then show ?thesis
    using  $\gamma(1,2)$  fresh-func[of nabra - t1 f]
    by simp
qed
qed

```

## 8 Equality

Proves various facts about the equivalence relation.

```

lemma equ-refl:
  shows nabra  $\vdash$  t  $\approx$  t
  by(induct t, auto simp add: ds-def)

```

```

lemma equ-dec-pi:
  assumes nabra  $\vdash$  swap pi t1  $\approx$  swap pi t2
  shows nabra  $\vdash$  t1  $\approx$  t2
proof -
  have i: nabra  $\vdash$  swap (rev pi) (swap pi t1)  $\approx$  t1
    nabra  $\vdash$  swap (rev pi) (swap pi t2)  $\approx$  t2
    using rev-pi-pi-equ by auto
  then have nabra  $\vdash$  swap (rev pi) (swap pi t1)  $\approx$  swap (rev pi) (swap pi t2)
    using equ-equivariance assms by simp
  then show ?thesis using i equ-symm equ-trans by meson
qed

```

```

lemma equ-involutive-left:
  shows nabra  $\vdash$  swap (rev pi) (swap pi t1)  $\approx$  t2 = nabra  $\vdash$  t1  $\approx$  t2
proof(auto)
  have i: nabra  $\vdash$  t1  $\approx$  swap (rev pi) (swap pi t1)
    using rev-pi-pi-equ equ-symm by blast
  show nabra  $\vdash$  swap (rev pi) (swap pi t1)  $\approx$  t2  $\implies$  nabra  $\vdash$  t1  $\approx$  t2
    using i equ-trans by blast
  show nabra  $\vdash$  t1  $\approx$  t2  $\implies$  nabra  $\vdash$  swap (rev pi) (swap pi t1)  $\approx$  t2
    using i equ-trans equ-symm by blast
qed

```

```

lemma equ-pi-to-left:

```

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

```
{assume i:  $\text{nabla} \vdash \text{swap } (\text{rev } \pi) \ t1 \approx t2$ 
have  $\text{nabla} \vdash \text{swap } \pi \ (\text{swap } (\text{rev } \pi) \ t1) \approx \text{swap } \pi \ t2$ 
  using equ-equivariance[OF i, of  $\pi$ ] by simp
then show  $\text{nabla} \vdash t1 \approx \text{swap } \pi \ t2$ 
  using equ-involutive-left[of  $\text{nabla} \ \langle \text{rev } \pi \rangle \ t1 \ \langle \text{swap } \pi \ t2 \rangle$ ] rev-rev-ident[of  $\pi$ ]
  by simp}
```

```
{assume i:  $\text{nabla} \vdash t1 \approx \text{swap } \pi \ t2$ 
have ii:  $\text{nabla} \vdash \text{swap } (\text{rev } \pi) \ t1 \approx \text{swap } (\text{rev } \pi) \ (\text{swap } \pi \ t2)$ 
  using equ-equivariance[OF i, of  $\langle \text{rev } \pi \rangle$ ] by simp
then have iii:  $\text{nabla} \vdash \text{swap } (\text{rev } \pi) \ (\text{swap } \pi \ t2) \approx \text{swap } (\text{rev } \pi) \ t1$ 
  using equ-symm[OF ii] by simp
then have iv:  $\text{nabla} \vdash t2 \approx \text{swap } (\text{rev } \pi) \ t1$ 
  using equ-involutive-left[of  $\text{nabla} \ \pi \ t2 \ \langle \text{swap } (\text{rev } \pi) \ t1 \rangle$ ] by simp
then show  $\text{nabla} \vdash \text{swap } (\text{rev } \pi) \ t1 \approx t2$ 
  using equ-symm[OF iv] by simp}
```

qed

**lemma equ- $\pi$ -to-right:**

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

**proof**

```
{assume i:  $\text{nabla} \vdash t1 \approx \text{swap } (\text{rev } \pi) \ t2$ 
  then show  $\text{nabla} \vdash \text{swap } \pi \ t1 \approx t2$ 
    using equ-involutive-left equ-dec- $\pi$  by blast}
{assume ii:  $\text{nabla} \vdash \text{swap } \pi \ t1 \approx t2$ 
  then show  $\text{nabla} \vdash t1 \approx \text{swap } (\text{rev } \pi) \ t2$ 
    using equ-involutive-left equ-equivariance by blast}
```

qed

**lemma equ-involutive-right:**

shows  $\text{nabla} \vdash t1 \approx \text{swap } (\text{rev } \pi) \ (\text{swap } \pi \ t2) = \text{nabla} \vdash t1 \approx t2$   
 using equ-dec- $\pi$ [of  $\text{nabla} \ \pi \ t1 \ t2$ ] equ-equivariance[of  $\text{nabla} \ t1 \ t2 \ \pi$ ]  
 equ- $\pi$ -to-right[of  $\text{nabla} \ t1 \ \pi \ \langle \text{swap } \pi \ t2 \rangle$ ]  
 by auto

**lemma equ- $\pi$ 1- $\pi$ 2-add:**

assumes  $\forall a \in ds \ \pi 1 \ \pi 2. \ \text{nabla} \vdash a \# t$   
 shows  $\text{nabla} \vdash \text{swap } \pi 1 \ t \approx \text{swap } \pi 2 \ t$

**proof** –

```
have  $\text{nabla} \vdash t \approx \text{swap } (\text{rev } \pi 1 \ @ \ \pi 2) \ t$ 
  using assms ds-rev equ- $\pi$ -right by simp
hence  $\text{nabla} \vdash t \approx \text{swap } (\text{rev } \pi 1) \ (\text{swap } \pi 2 \ t)$ 
  using swap-append by auto
then show  $\text{nabla} \vdash \text{swap } \pi 1 \ t \approx \text{swap } \pi 2 \ t$ 
  using equ- $\pi$ -to-right by simp
```

qed

**lemma** *pi-right-equ*:

**assumes**  $nabla \vdash t \approx \text{swap } \pi \ t$   
**shows**  $\forall a \in ds \ [] \ \pi. \ nabla \vdash a \ \# \ t$   
**using** *assms pi-right-equ-help* **by** *blast*

**lemma** *equ-pi1-pi2-dec*:

**assumes**  $nabla \vdash \text{swap } \pi_1 \ t \approx \text{swap } \pi_2 \ t$   
**shows**  $\forall a \in ds \ \pi_1 \ \pi_2. \ nabla \vdash a \ \# \ t$

**proof** –

**have**  $nabla \vdash t \approx \text{swap } ((\text{rev } \pi_1) \ @ \ \pi_2) \ t$   
**using** *assms equ-pi-to-right swap-append* **by** *simp*  
**then show**  $\forall a \in ds \ \pi_1 \ \pi_2. \ nabla \vdash a \ \# \ t$   
**using** *pi-right-equ ds-rev*[of  $\pi_1 \ \pi_2$ ] **by** *simp*

qed

**lemma** *equ-weak*:

**assumes**  $nabla_1 \vdash t_1 \approx t_2$   
**shows**  $(nabla_1 \cup nabla_2) \vdash t_1 \approx t_2$   
**using** *assms* **by** (*erule-tac equ.induct*) (*auto simp add: fresh-weak*)

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

**lemma** *psub-trm-not-equ*:

**shows**  $\forall t_2 \in \text{psub-trms } t_1. \ (\neg(\exists \pi. (nabla \vdash t_1 \approx \text{swap } \pi \ t_2)))$

**proof**

**fix**  $t_2$

**assume**  $i: t_2 \in \text{psub-trms } t_1$

**show**  $\neg(\exists \pi. nabla \vdash t_1 \approx \text{swap } \pi \ t_2)$

**proof**

**assume**  $\exists \pi. nabla \vdash t_1 \approx \text{swap } \pi \ t_2$

**then obtain**  $\pi$  **where**  $H$ :

$nabla \vdash t_1 \approx \text{swap } \pi \ t_2$  **by** *blast*

**from** *equ-depth*[OF  $H$ ]

**have**  $\text{depth } t_1 = \text{depth } (\text{swap } \pi \ t_2)$

**by** *simp*

**hence**  $\text{depth } t_1 = \text{depth } t_2$

**by** *simp*

**moreover have**  $\text{depth } t_2 < \text{depth } t_1$

**using**  $i$  *depth-psub-trms*

**by** *simp*

**ultimately show** *False* **by** *auto*

qed

qed

## 9 Substitutions

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

**type-synonym** *subst* = (*string* × *trm*) *list*

**fun** *look-up* :: *string* ⇒ *subst* ⇒ *trm* **where**

*look-up* *X* [] = *Susp* [] *X* |

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

**fun** *subst* :: *subst* ⇒ *trm* ⇒ *trm* **where**

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

*subst-susp*: *subst* *s* (*Susp* *pi* *X*) = *swap* *pi* (*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* :: *subst* ⇒ *subst* ⇒ (*string* ⇒ *trm* ⇒ *subst* ⇒ *subst* ⇒ *subst*) ⇒ *subst* **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* :: *subst* ⇒ *subst* ⇒ *subst* (**infixl** <·> 81) **where**

*s1* · *s2* ≡ *alist-rec* *s2* *s1* (λ *x* *y* *xs* *g*. (*x*, *subst* *s1* *y*) # *g*)

The domain of substitutions.

**definition** *domn* :: (*trm* ⇒ *trm*) ⇒ *string* *set* **where**

*domn* *s* ≡ {*X*. (*s* (*Susp* [] *X*)) ≠ (*Susp* [] *X*)}

substitutions extend freshness environments.

**definition** *ext-subst* :: *fresh-envs* ⇒ (*trm* ⇒ *trm*) ⇒ *fresh-envs* ⇒ *bool* ( - ⊨ - - [80,80,80] 80) **where**

*nabla1* ⊨ *s* *nabla2* ≡ (∀ (*a*, *X*) ∈ *nabla2*. *nabla1* ⊢ *a* # *s* (*Susp* [] *X*))

Alpha-equality for substitutions

**definition** *subst-equ* :: *fresh-envs* ⇒ (*trm* ⇒ *trm*) ⇒ (*trm* ⇒ *trm*) ⇒ *bool* ( - ⊨ - ≈ - [80,80,80] 80)

**where**

*nabla* ⊨ *s1* ≈ *s2* ≡ ∀ *X* ∈ (*domn* *s1* ∪ *domn* *s2*). (*nabla* ⊢ *s1* (*Susp* [] *X*) ≈ *s2* (*Susp* [] *X*))

**lemma** *subst-equ-symm*:

```

assumes  $nabla \models s1 \approx s2$ 
shows  $nabla \models s2 \approx s1$ 
using assms equ-symm subst-eq-def[of  $nabla s1 s2$ ]
      subst-eq-def[of  $nabla s2 s1$ ] Un-commute[of  $\langle \text{domn } s1 \rangle \langle \text{domn } s2 \rangle$ ] by blast

lemma subst-eq-refl:
shows  $nabla \models s \approx s$ 
using eq-refl subst-eq-def Un-absorb by simp

lemma not-in-domn:
assumes  $X \notin (\text{domn } s)$ 
shows  $s (\text{Susp } [] X) = (\text{Susp } [] X)$ 
using assms domn-def by simp

lemma subst-swap-comm:
shows  $\text{subst } s (\text{swap } pi \ t) = \text{swap } pi (\text{subst } s \ t)$ 
using swap-append subst-susp by (induct t) auto

lemma subst-not-occurs:
assumes  $\neg(\text{occurs } X \ t)$ 
shows  $\text{subst } [(X, t2)] \ t = t$ 
proof –
  have  $i: \neg(\text{occurs } X \ t) \longrightarrow \text{subst } [(X, t2)] \ t = t$ 
    using subst-susp by (induct t) auto
  then show ?thesis using assms by simp
qed

lemma id-subst [simp]:
shows  $\text{subst } [] \ t = t$ 
using subst-susp by (induct t) auto

lemma subst-comp-id [simp]:
shows  $\text{subst } (s \cdot []) = \text{subst } s$ 
using comp-def by simp

lemma id-comp-subst [simp]:
shows  $\text{subst } ([] \cdot s) = \text{subst } s$ 
proof(rule ext)
  fix  $t$ 
  show  $\text{subst } ([] \cdot s) \ t = \text{subst } s \ t$ 
  proof(induction t arbitrary: s)
    case (Susp pi X)
    then show ?case using comp-def subst-susp
    proof (induction s)
      case Nil
      then show ?case
      using comp-def subst-susp by simp
    next
    case (Cons a s)

```

```

    then show ?case
      using comp-def subst-susp by simp
    qed
  qed (simp-all)
qed

lemma subst-comp-expand:
  shows subst (s1 · s2) t = subst s1 (subst s2 t)
proof(induct t)
  case (Susp x1 x2)
  then show ?case
  proof(induct s2)
    case Nil
    then show ?case using comp-def subst-susp subst-swap-comm by simp
  next
    case (Cons a s2)
    then show ?case using comp-def subst-susp subst-swap-comm by simp
  qed
qed (simp-all)

lemma subst-assoc:
  shows subst (s1 · (s2 · s3)) = subst ((s1 · s2) · s3)
proof(rule ext)
  fix t
  show subst (s1 · (s2 · s3)) t = subst (s1 · s2 · s3) t
  proof(induct t)
    case (Susp pi X)
    then show ?case using subst-comp-expand by simp
  qed (simp-all)
qed

lemma fresh-subst:
  assumes nabra1 ⊢ a # t
    and nabra2 ⊨ (subst s) nabra1
  shows nabra2 ⊢ a # subst s t
  using assms
proof(induction rule: fresh.induct)
  case (fresh-susp pi a X nabra)
  then show ?case
  using ext-subst-def subst-susp fresh-swap-right
  by auto
qed (auto)

lemma equ-subst:
  assumes nabra1 ⊢ t1 ≈ t2
    and nabra2 ⊨ (subst s) nabra1
  shows nabra2 ⊢ (subst s t1) ≈ (subst s t2)
  using assms
proof(induction rule: equ.induct)

```

```

case (equ-abst-ab a b nabla t2 t1)
then show ?case
  using subst-swap-comm fresh-susb equ.equ-abst-ab by simp
next
case (equ-susp pi1 pi2 X nabla)
then show ?case
  using subst-susp equ-pi1-pi2-add ext-susb-def subst-susp
  by fastforce
qed (auto)

lemma unif-1:
assumes nabla ⊢ subst s (Susp pi X) ≈ subst s t
shows nabla ⊨ subst (s · [(X, swap (rev pi) t)]) ≈ subst s
using assms
apply(simp only: subst-equ-def)
proof
fix Xa
assume hyps: nabla ⊢ subst s (Susp pi X) ≈ subst s t
  Xa ∈ domn (subst (s · [(X, swap (rev pi) t)]) ∪ domn (subst s))
show nabla ⊢ subst (s · [(X, swap (rev pi) t)]) (Susp [] Xa) ≈ subst s (Susp []
Xa)
proof–
have one:
  nabla ⊢ subst (s · [(X, swap (rev pi) t)]) (Susp pi Xa) ≈
  subst s (subst [(X, swap (rev pi) t)] (Susp pi Xa))
  using subst-comp-expand equ-refl by simp
have two:
  nabla ⊢ subst s (subst [(X, swap (rev pi) t)] (Susp pi Xa)) ≈
  subst s (swap pi (look-up Xa [(X, swap (rev pi) t)]))
  using equ-refl subst-susp[of ⟨[(X, swap (rev pi) t)]⟩ pi Xa]
  by simp
have nabla ⊢ subst (s · [(X, swap (rev pi) t)]) (Susp pi Xa) ≈
  subst s (swap pi (look-up Xa [(X, swap (rev pi) t)]))
  using equ-trans[OF one two] by simp
hence nabla ⊢ swap pi (look-up Xa (s · [(X, swap (rev pi) t)])) ≈
  subst s (swap pi (look-up Xa [(X, swap (rev pi) t)]))
  using subst-susp[of ⟨(s · [(X, swap (rev pi) t)])⟩] by simp
hence swap-pi-outside: nabla ⊢ swap pi (look-up Xa (s · [(X, swap (rev pi)
t)])) ≈
  swap pi (subst s (look-up Xa [(X, swap (rev pi) t)]))
  using subst-swap-comm by simp
hence nabla ⊢ (look-up Xa (s · [(X, swap (rev pi) t)])) ≈
  (subst s (look-up Xa [(X, swap (rev pi) t)]))
  using equ-dec-pi[OF swap-pi-outside] by simp
hence three:
  nabla ⊢ subst (s · [(X, swap (rev pi) t)]) (Susp [] Xa) ≈
  (subst s (look-up Xa [(X, swap (rev pi) t)]))
  using subst-susp by simp
then show ?thesis

```

```

proof(cases  $Xa = X$ )
  case True
    hence  $\text{subst } s \text{ (look-up } Xa \text{ [(X, swap (rev pi) t)])} = \text{subst } s \text{ (swap (rev pi) t)}$ 
      by simp
    with three have
       $\text{nabla} \vdash \text{subst } (s \cdot [(X, \text{swap (rev pi) t})]) \text{ (Susp [] } Xa) \approx \text{swap (rev pi)}$ 
      ( $\text{subst } s \text{ t}$ )
      using subst-swap-comm by auto
    hence  $i: \text{nabla} \vdash \text{swap pi (subst } (s \cdot [(X, \text{swap (rev pi) t)]) \text{ (Susp [] } Xa))$ 
       $\approx \text{subst } s \text{ t}$  using swap-inv-side by simp
    hence  $\text{nabla} \vdash \text{swap pi (subst } (s \cdot [(X, \text{swap (rev pi) t)]) \text{ (Susp [] } Xa))$ 
       $\approx \text{subst } s \text{ (Susp pi } Xa)$ 
      using True equ-trans[OF i equ-symm[OF assms(1)]] by simp
    hence  $\text{nabla} \vdash \text{swap pi (subst } (s \cdot [(X, \text{swap (rev pi) t)]) \text{ (Susp [] } Xa))$ 
       $\approx \text{swap pi (look-up } Xa \text{ s)}$ 
      using subst-susp by simp
    then show  $\text{nabla} \vdash (\text{subst } (s \cdot [(X, \text{swap (rev pi) t)]) \text{ (Susp [] } Xa))$ 
       $\approx \text{subst } s \text{ (Susp [] } Xa)$ 
      using equ-dec-pi subst-susp by simp
  next
    case False
      hence  $\text{subst } s \text{ (look-up } Xa \text{ [(X, swap (rev pi) t)])} = \text{subst } s \text{ (Susp [] } Xa)$ 
        by simp
      with three show ?thesis by auto
    qed
  qed
qed

```

```

lemma subst-equ-a:
  assumes  $\text{nabla} \models (\text{subst } s1) \approx (\text{subst } s2)$ 
    and  $\text{nabla} \vdash (\text{subst } s2 \text{ t1}) \approx \text{t2}$ 
  shows  $\text{nabla} \vdash (\text{subst } s1 \text{ t1}) \approx \text{t2}$ 
  using assms
proof(induct t1 arbitrary: t2)
  case (Abst a t1')
    then obtain  $t2' \ b$ 
      where  $t2: t2 = \text{Abst } b \text{ t2'}$ 
      by(auto elim: equ.cases)
    with Abst(3) have
       $i: \text{nabla} \vdash \text{Abst } a \text{ (subst } s2 \text{ t1')} \approx \text{Abst } b \text{ t2'}$ 
      using subst-abst by simp
    then show ?case
  proof(cases a=b)
    case True
      hence  $\text{nabla} \vdash (\text{subst } s2 \text{ t1}') \approx \text{t2'}$ 
        using Equ-elim(7)[OF i] by auto
      with Abst(1)[OF Abst(2), of t2']
      have  $\text{nabla} \vdash \text{subst } s1 \text{ t1}' \approx \text{t2'}$ 
        by simp

```

**hence**  $nabla \vdash \text{Abst } a \text{ (subst } s1 \text{ } t1') \approx \text{Abst } b \text{ } t2'$   
**using** *True equ-abst-aa* **by** *simp*  
**with**  $t2$  **show** *?thesis*  
**using** *subst-abst[of s1 a t1']* **by** *simp*  
**next**  
**case** *False*  
**hence**  $nabla \vdash (\text{subst } s2 \text{ } t1') \approx \text{swap } [(a,b)] \text{ } t2'$   
**and** *fresh: nabla \vdash a \# t2'*  
**using** *Equ-elim(7)[OF i]* **by** *auto*  
**with** *Abst(1)[OF Abst(2), of \langle swap [(a,b)] t2' \rangle]*  
**have**  $nabla \vdash \text{subst } s1 \text{ } t1' \approx \text{swap } [(a,b)] \text{ } t2'$   
**by** *simp*  
**hence**  $nabla \vdash \text{Abst } a \text{ (subst } s1 \text{ } t1') \approx \text{Abst } b \text{ } t2'$   
**using** *equ-abst-ab[OF False fresh]* **by** *simp*  
**with**  $t2$  **show** *?thesis*  
**using** *subst-abst[of s1 a t1']* **by** *simp*  
**qed**  
**next**  
**case** *(Susp pi X)*  
**hence**  $nabla \vdash \text{swap } pi \text{ (look-up } X \text{ } s2) \approx t2$   
**using** *subst-susp* **by** *simp*  
**hence**  $nabla \vdash (\text{look-up } X \text{ } s2) \approx \text{swap } (\text{rev } pi) \text{ } t2$   
**using** *subst-susp swap-inv-side* **by** *simp*  
**hence**  $i: nabla \vdash \text{subst } s2 \text{ (Susp } [] \text{ } X) \approx \text{swap } (\text{rev } pi) \text{ } t2$   
**using** *subst-susp[of s2 \langle [] \rangle]* **by** *simp*  
  
**with** *Susp(1) subst-equ-def[of nabla \langle subst s1 \rangle \langle subst s2 \rangle]*  
**have**  $nabla \vdash \text{subst } s1 \text{ (Susp } [] \text{ } X) \approx \text{subst } s2 \text{ (Susp } [] \text{ } X)$   
**using** *UnCI equ-refl not-in-donn* **by** *metis*  
  
**with**  $i$  **have**  $nabla \vdash \text{subst } s1 \text{ (Susp } [] \text{ } X) \approx \text{swap } (\text{rev } pi) \text{ } t2$   
**using** *equ-trans* **by** *blast*  
**hence**  $nabla \vdash (\text{look-up } X \text{ } s1) \approx \text{swap } (\text{rev } pi) \text{ } t2$   
**using** *subst-susp* **by** *simp*  
**hence**  $nabla \vdash \text{swap } pi \text{ (look-up } X \text{ } s1) \approx t2$   
**using** *swap-inv-side* **by** *simp*  
**then** **show** *?case*  
**using** *subst-susp* **by** *simp*  
**next**  
**case** *(Paar t11 t12)*  
**then** **obtain**  $t21 \text{ } t22$   
**where**  $t2: t2 = \text{Paar } t21 \text{ } t22$   
**by** *(auto elim: equ.cases)*  
**with** *Paar(4)*  
**have**  $\text{paar-}i: nabla \vdash \text{subst } s2 \text{ } t11 \approx t21$   
**and**  $\text{paar-}ii: nabla \vdash \text{subst } s2 \text{ } t12 \approx t22$   
**by** *(auto elim: equ.cases)*  
**from**  $t2$  **show** *?case*  
**using** *equ-paar[OF Paar(1)[OF Paar(3) paar-i] Paar(2)[OF Paar(3) paar-ii]]*

```

      subst-paar by simp
next
case (Func f t1')
then obtain t2'
  where t2: t2 = Func f t2'
  by(auto elim: equ.cases)
  with Func(3)
  have func: nablā ⊢ subst s2 t1' ≈ t2'
  by (auto elim: equ.cases)
  from t2 show ?case
  using equ-func[OF Func(1)[OF Func(2) func], of f] by simp
qed (simp-all)

lemma unif-2a:
  assumes nablā ⊨ subst s1 ≈ subst s2
  and (nablā ⊢ subst s2 t1 ≈ subst s2 t2)
shows (nablā ⊢ subst s1 t1 ≈ subst s1 t2)
proof -
  have i: (nablā ⊢ subst s1 t1 ≈ subst s2 t2)
  using subst-equ-a[OF assms(1,2)] by simp
  show (nablā ⊢ subst s1 t1 ≈ subst s1 t2)
  using subst-equ-a[OF assms(1) equ-symm[OF i]] equ-symm by auto
qed

lemma unif-2b:
  assumes nablā ⊨ subst s1 ≈ subst s2
  and nablā ⊢ a ‡ subst s2 t
shows nablā ⊢ a ‡ subst s1 t
  using assms
proof(induct t)
  case (Abst b t')
  then show ?case
  proof(cases a=b)
    case True
    then show ?thesis
    using fresh-abst-aa by simp
  next
  case False
  with Abst show ?thesis
  using Fresh-elim(1) subst-abst fresh-abst-ab
  by metis
qed
next
case (Susp pi X)
then show ?case
  using equ-refl equ-symm l3-jud subst-equ-a by meson
next
case (Paar t1 t2)
then show ?case

```

```

    using Fresh-elim5 subst-paar fresh-paar
    by metis
next
case (Func f t')
then show ?case
    using Fresh-elim6 subst-func fresh-func
    by metis
qed (simp-all)

```

```

lemma subst-equ-to-trm:
  assumes nabl1  $\models$  subst s1  $\approx$  subst s2
  shows nabl1  $\vdash$  subst s1 t  $\approx$  subst s2 t
  using assms equ-refl subst-equ-a by simp

```

```

lemma subst-cancel-right:
  assumes nabl1  $\models$  (subst s1)  $\approx$  (subst s2)
  shows nabl1  $\models$  (subst (s1  $\cdot$  s))  $\approx$  (subst (s2  $\cdot$  s))
  using assms subst-comp-expand subst-equ-def subst-equ-to-trm
  by simp

```

```

lemma subst-trans:
  assumes nabl1  $\models$  subst s1  $\approx$  subst s2 nabl2  $\models$  subst s2  $\approx$  subst s3
  shows nabl1  $\models$  subst s1  $\approx$  subst s3
  using assms equ-trans subst-equ-def subst-equ-to-trm by meson

```

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

```

lemma occurs-sub-trm-equ:
  assumes occurs X t1
  shows  $\exists t2 \in \text{sub-trms } (\text{subst } s \ t1). (\exists pi. (\text{nabl1} \vdash \text{subst } s \ (\text{Susp } pi1 \ X) \approx (\text{swap } pi \ t2)))$ 
  using assms
proof (induct t1)
  case (Susp pi' X)
  then show ?case
    using equ-pi-to-left equ-involutive-left
    equ-refl subst-susp swap-append t-sub-trms-t
    occurs.simps(3) by metis
next
  case (Paar t11 t12)
  then show ?case
    using subst-paar Un-iff occurs.simps(5) psub-sub-trms
    psub-trms.simps(4) by (metis)
qed (auto)

```

```

lemma ext-subst-strong:
  assumes nabl1  $\models$  (subst s) (nabl2  $\cup$  nabl3)
  shows nabl1  $\models$  (subst s) nabl2 and nabl1  $\models$  (subst s) nabl3
  using assms
  unfolding ext-subst-def by auto

```

**lemma** *ext-subst-id*:  
**shows**  $nabla \models (subst \ []) nabla$   
**unfolding** *ext-subst-def id-subst* **by** *auto*

## 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-type* =  $((trm \times trm) list) \times ((string \times trm) list)$

**type-synonym** *unifier-type* =  $fresh-envs \times substs$

**definition** *U* ::  $problem-type \Rightarrow (unifier-type set)$

**where** *all-solutions-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** *eprobs* =  $((trm \times trm) list)$

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

**type-synonym** *probs* =  $eprobs \times fprobs$

**fun** *vars-fprobs* ::  $((string \times trm) list) \Rightarrow (string set)$

**where**

$vars-fprobs \ [] = \{\}$  |  
 $vars-fprobs (x\#xs) = (vars-trm (snd x)) \cup (vars-fprobs xs)$

**fun** *vars-eprobs* ::  $((trm \times trm) list) \Rightarrow (string set)$

**where**

$vars-eprobs \ [] = \{\}$  |  
 $vars-eprobs (x\#xs) = (vars-trm (snd x)) \cup (vars-trm (fst x)) \cup (vars-eprobs xs)$

**definition** *vars-probs* ::  $problem-type \Rightarrow nat$

**where**  $vars-probs P \equiv card((vars-fprobs (snd P)) \cup (vars-eprobs (fst P)))$

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

**definition**  $apply\text{-}subst\text{-}fprobs :: substs \Rightarrow fprobs \Rightarrow fprobs$   
**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**  
 $unit\text{-}sred[intro!]: ((Unit \approx? Unit) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys) \mid$   
 $paar\text{-}sred[intro!]: ((Paar t1 t2 \approx? Paar s1 s2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? s1) \# (t2 \approx? s2) \# xs, ys)$   
 $\mid$   
 $func\text{-}sred[intro!]: ((Func F t1 \approx? Func F t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys) \mid$   
 $abst\text{-}aa\text{-}sred[intro!]: ((Abst a t1 \approx? Abst a t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys) \mid$   
 $abst\text{-}ab\text{-}sred[intro!]: a \neq b \implies$   
 $((Abst a t1 \approx? Abst b t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? swap [(a, b)]$   
 $t2) \# xs, (a \#? t2) \# ys) \mid$   
 $atom\text{-}sred[intro!]: ((Atom a \approx? Atom a) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys) \mid$   
 $susp\text{-}sred[intro!]: ((Susp pi1 X \approx? Susp pi2 X) \# xs, ys)$   
 $\vdash [] \rightsquigarrow (xs, (map (\lambda a. a \#? Susp [] X) (ds\text{-}list pi1 pi2))) @ ys) \mid$   
 $var\text{-}1\text{-}sred[intro!]: \neg(occurs X t) \implies ((Susp pi X \approx? t) \# xs, ys)$   
 $\vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-}subst [(X, swap (rev pi) t)]$   
 $(xs, ys) \mid$   
 $var\text{-}2\text{-}sred[intro!]: \neg(occurs X t) \implies ((t \approx? Susp pi X) \# xs, ys)$   
 $\vdash [(X, swap (rev pi) t)] \rightsquigarrow apply\text{-}subst [(X, swap (rev pi) t)]$   
 $(xs, ys)$

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

$((Unit \approx? Unit) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$   
 $((Paar t1 t2 \approx? Paar s1 s2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? s1) \# (t2 \approx? s2) \# xs, ys)$   
 $((Func F t1 \approx? Func F t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys)$   
 $((Abst a t1 \approx? Abst a t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? t2) \# xs, ys)$   
 $((Abst a t1 \approx? Abst b t2) \# xs, ys) \vdash [] \rightsquigarrow ((t1 \approx? swap [(a, b)] t2) \# xs, (a \#? t2) \# ys)$   
 $((Atom a \approx? Atom a) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$

$((\text{Susp } \pi 1 \ X \approx ? \text{Susp } \pi 2 \ X) \# xs, ys) \vdash [] \rightsquigarrow (xs, (\text{map } (\lambda a. a \# ? \text{Susp } [] \ X) (\text{ds-list } \pi 1 \ \pi 2))) @ ys)$   
 $((\text{Susp } \pi \ X \approx ? t) \# xs, ys) \vdash [(X, \text{swap } (\text{rev } \pi) \ t)] \rightsquigarrow \text{apply-subst } [(X, \text{swap } (\text{rev } \pi) \ t)] (xs, ys)$   
 $((t \approx ? \text{Susp } \pi \ X) \# xs, ys) \vdash [(X, \text{swap } (\text{rev } \pi) \ t)] \rightsquigarrow \text{apply-subst } [(X, \text{swap } (\text{rev } \pi) \ t)] (xs, ys)$

**lemma** *sred-symm*:

**assumes**  $((t1 \approx ? t2) \# xs, ys) \vdash s \rightsquigarrow P2$   
**shows**  $\exists P3. ((t2 \approx ? t1) \# xs, ys) \vdash s \rightsquigarrow P3$   
**using** *assms*

**proof** (*cases rule: s-red.cases*)

**case** (*var-1-sred*  $X \ \pi$ )

**have**  $((t2, \text{Susp } \pi \ X) \# xs,$

$ys) \vdash [(X, \text{swap } (\text{rev } \pi) \ t2)] \rightsquigarrow \text{apply-subst } [(X, \text{swap } (\text{rev } \pi) \ t2)] (xs, ys)$

**using** *var-2-sred* [*OF*  $\langle \neg \text{occurs } X \ t2 \rangle$ ] **by** *simp*

**then show** *?thesis* **using** *var-1-sred* **by** *blast*

**next**

**case** (*var-2-sred*  $X \ \pi$ )

**have**  $((\text{Susp } \pi \ X, t1) \# xs,$

$ys) \vdash [(X, \text{swap } (\text{rev } \pi) \ t1)] \rightsquigarrow \text{apply-subst } [(X, \text{swap } (\text{rev } \pi) \ t1)] (xs, ys)$

**using** *var-1-sred* [*OF*  $\langle \neg \text{occurs } X \ t1 \rangle$ ] **by** *simp*

**then show** *?thesis* **using** *var-2-sred* **by** *blast*

**qed** (*auto*)

Weakening of freshness

**lemma** *solution-weak*:

**assumes**  $(\text{nabla}1, s) \in U \ P$

**shows**  $(\text{nabla}1 \cup \text{nabla}3, s) \in U \ P$

**using** *assms*

**unfolding** *all-solutions-def* **using** *fresh-weak equ-weak* **by** *auto*

**lemma** *solution-comp-id*:

**shows**  $((\text{nabla}, s) \in U \ P) = ((\text{nabla}, s \cdot []) \in U \ P)$  **and**

$((\text{nabla}, s) \in U \ P) = ((\text{nabla}, [] \cdot s) \in U \ P)$

**unfolding** *all-solutions-def* **by** *auto*

**lemma** *solutions-subst-assoc*:

$((\text{nabla}, s1 \cdot (s2 \cdot s3)) \in U \ P) = ((\text{nabla}, (s1 \cdot s2) \cdot s3) \in U \ P)$

**unfolding** *all-solutions-def* **using** *subst-assoc* **by** *simp*

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!*]:  $([], (a \# ? \text{Unit}) \# xs) \vdash \{\} \rightarrow ([], xs) \mid$

*paar-cred*[*intro!*]:  $([], (a \# ? \text{Paar } t1 \ t2) \# xs) \vdash \{\} \rightarrow ([], (a \# ? t1) \# (a \# ? t2) \# xs) \mid$

*func-cred*[*intro!*]:  $([], (a \# ? \text{Func } F \ t) \# xs) \vdash \{\} \rightarrow ([], (a \# ? t) \# xs) \mid$

$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**  $fst\ P1 = []$

**using** *assms* **by** (*auto elim: c-red.cases*)

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**  $\exists nabla1\ s1\ P3. ((t2 \approx? t1) \# xs, ys) \models (nabla1, s1) \Rightarrow P3$

**using** *assms*

**proof** (*induction arbitrary: nabla s rule: red-plus.induct[OF assms(2)]*)

**case** ( $1\ P1\ s1\ P$ )

**hence**  $((t1, t2) \# xs, ys) \vdash s1 \rightsquigarrow P$  **by** *simp*

**hence**  $\exists P3. ((t2 \approx? t1) \# xs, ys) \vdash s1 \rightsquigarrow P3$

**using** *sred-symm* **by** *simp*

**hence**  $\exists P3. ((t2 \approx? t1) \# xs, ys) \models (\{\}, s1) \Rightarrow P3$

**using** *sred-single* **by** *auto*

**then show**  $\exists nabla1\ s1\ P3. ((t2, t1) \# xs, ys) \models (nabla1, s1) \Rightarrow P3$  **by** *auto*

**next**

**case** ( $2\ P1\ nabla1\ P2$ )

**hence**  $fst\ P1 \neq []$  **by** *simp*

**moreover from**  $2(1)$  **have**  $fst\ P1 = []$

**using** *c-red-eqs-empty* **by** *simp*

**ultimately have** *False* **by** *simp*

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

**next**

**case** ( $3\ P1\ s1\ P'\ nabla2\ s2\ P3$ )

**hence**  $((t1, t2) \# xs, ys) \vdash s1 \rightsquigarrow P'$  **by** *simp*

**hence**  $\exists P3. ((t2 \approx? t1) \# xs, ys) \vdash s1 \rightsquigarrow P3$

**using** *sred-symm* **by** *simp*

**hence**  $\exists P3. ((t2 \approx? t1) \# xs, ys) \models (\{\}, s1) \Rightarrow P3$

**using** *sred-single* **by** *auto*

**then show**  $\exists \text{nabla}1 \ s1 \ P3. ((t2, t1) \# \text{xs}, \text{ys}) \models (\text{nabla}1, s1) \Rightarrow P3$  **by auto**  
**next**  
**case** ( $\_4 \ P1 \ \text{nabla}1 \ P' \ \text{nabla}2 \ P3$ )  
**hence**  $\text{fst } P1 \neq []$  **by simp**  
**moreover from**  $\_4(1)$  **have**  $\text{fst } P1 = []$   
**using** *c-red-eqs-empty* **by simp**  
**ultimately have** *False* **by simp**  
**then show** *?case* **by simp**  
**qed**

**lemma** *mgu-idem*:

**assumes**  $(\text{nabla}1, s1) \in U \ P$   
**and**  $\forall (\text{nabla}2, s2) \in U \ P. \ \text{nabla}2 \models (\text{subst } s2) \ \text{nabla}1 \ \wedge \ \text{nabla}2 \models \text{subst}(s2 \cdot s1) \approx \text{subst } s2$   
**shows**  $\text{mgu } P \ (\text{nabla}1, s1) \ \wedge \ \text{idem } (\text{nabla}1, s1)$   
**using** *assms mgu-def idem-def* **by auto**

**lemma** *problem-subst-comm*:

**shows**  $((\text{nabla}, s2) \in U \ (\text{apply-subst } s1 \ P)) = ((\text{nabla}, (s2 \cdot s1)) \in U \ P)$   
**using** *all-solutions-def apply-subst-def subst-comp-expand* **by auto**

Preservation of solutions

**lemma** *P1-to-P2-sred*:

**assumes**  $(\text{nabla}1, s1) \in U \ P1$  **and**  $P1 \vdash_{s2} \rightsquigarrow P2$   
**shows**  $(\text{nabla}1, s1) \in U \ P2 \ \wedge \ (\text{nabla}1 \models \text{subst} \ (s1 \cdot s2) \approx \text{subst } s1)$   
**using** *assms*  
**proof**(*induction arbitrary: nabla1 s1 rule: s-red.induct[OF (P1  $\vdash_{s2}$   $\rightsquigarrow$  P2)]*)  
**case** ( $1 \ \text{xs} \ \text{ys}$ )  
**then have**  $\text{subst}: \text{nabla}1 \models \text{subst} \ (s1 \cdot []) \approx \text{subst } s1$   
**using** *equ-refl subst-equ-def* **by simp**  
**from**  $1$   
**have eqs:**  $\forall (t1, t2) \in \text{set } \text{xs}. \ \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**and fresh:**  $\forall (a, t) \in \text{set } \text{ys}. \ \text{nabla}1 \vdash a \ \# \ \text{subst } s1 \ t$   
**using** *all-solutions-def* **by simp-all**  
**with** *subst* **show** *?case*  
**using** *all-solutions-def* **by auto**  
**next**  
**case** ( $2 \ t1 \ t2 \ t3 \ t4 \ \text{xs} \ \text{ys}$ )  
**then have**  $\text{subst}: \text{nabla}1 \models \text{subst} \ (s1 \cdot []) \approx \text{subst } s1$   
**using** *equ-refl subst-equ-def* **by simp**  
  
**from**  $2$   
**have fresh:**  $\forall (a, t) \in \text{set } \text{ys}. \ \text{nabla}1 \vdash a \ \# \ \text{subst } s1 \ t$   
**using** *all-solutions-def* **by simp**  
  
**from**  $2$   
**have**  $i: \forall (t1, t2) \in \text{set} \ ((\text{Paar } t1 \ t2, \ \text{Paar } t3 \ t4) \# \ \text{xs}). \ \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$

```

    using all-solutions-def by simp
  hence  $nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t3$ 
     $nabla1 \vdash \text{subst } s1 \ t2 \approx \text{subst } s1 \ t4$ 
    using Equ-elim3(4)[of  $nabla1 \ \langle \text{subst } s1 \ t1 \rangle \ \langle \text{subst } s1 \ t2 \rangle \ \langle \text{subst } s1 \ t3 \rangle$ 
 $\langle \text{subst } s1 \ t4 \rangle$ ]
      subst-paar[of  $s1$ ] by auto+
    with  $i$  have eqs:
       $\forall (t1, t2) \in \text{set } ((t1, t3) \# (t2, t4) \# xs). \ \nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
      by auto

  from fresh eqs subst
  show ?case using all-solutions-def by simp
next
case ( $\exists F \ t1 \ t2 \ xs \ ys$ )
then have subst:  $nabla1 \models \text{subst } (s1 \cdot []) \approx \text{subst } s1$ 
  using equ-refl subst-eq-def by simp

  from  $\exists$ 
  have fresh:  $\forall (a, t) \in \text{set } ys. \ \nabla1 \vdash a \ \# \ \text{subst } s1 \ t$ 
    using all-solutions-def by simp

  from  $\exists$ 
  have  $i$ :  $\forall (t1, t2) \in \text{set } ((\text{Func } F \ t1, \ \text{Func } F \ t2) \# xs). \ \nabla1 \vdash \text{subst } s1 \ t1 \approx$ 
 $\text{subst } s1 \ t2$ 
    using all-solutions-def by simp
  hence  $nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
    using Equ-elim3(5)[of  $nabla1 \ F \ \langle \text{subst } s1 \ t1 \rangle \ \langle \text{subst } s1 \ t2 \rangle$ ]
      subst-func by auto
  with  $i$  have eqs:
     $\forall (t1, t2) \in \text{set } ((t1, t2) \# xs). \ \nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
    by auto

  from subst fresh eqs
  show ?case using all-solutions-def by simp
next
case ( $\exists a \ t1 \ t2 \ xs \ ys$ )
then have subst:  $nabla1 \models \text{subst } (s1 \cdot []) \approx \text{subst } s1$ 
  using equ-refl subst-eq-def by simp

  from  $\exists$ 
  have fresh:  $\forall (a, t) \in \text{set } ys. \ \nabla1 \vdash a \ \# \ \text{subst } s1 \ t$ 
    using all-solutions-def by simp

  from  $\exists$ 
  have  $i$ :  $\forall (t1, t2) \in \text{set } ((\text{Abst } a \ t1, \ \text{Abst } a \ t2) \# xs). \ \nabla1 \vdash \text{subst } s1 \ t1 \approx$ 
 $\text{subst } s1 \ t2$ 
    using all-solutions-def by simp
  hence  $nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
    using Equ-elim3(6)[of  $nabla1 \ a \ \langle \text{subst } s1 \ t1 \rangle \ \langle \text{subst } s1 \ t2 \rangle$ ]

```

```

    subst-abst by auto
with i have eqs:
   $\forall (t1,t2) \in \text{set } ((t1, t2) \# xs). \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
  by auto

from subst fresh eqs
show ?case using all-solutions-def by simp
next
case (5 a b t1 t2 xs ys)
then have subst:  $\text{nabla}1 \models \text{subst } (s1 \cdot []) \approx \text{subst } s1$ 
  using equ-refl subst-equ-def by simp

from 5
have eqsP1:  $\forall (t1,t2) \in \text{set } ((\text{Abst } a \ t1, \text{Abst } b \ t2) \# xs). \text{nabla}1 \vdash \text{subst } s1 \ t1$ 
 $\approx \text{subst } s1 \ t2$ 
  using all-solutions-def by simp
hence i:  $\text{nabla}1 \vdash \text{Abst } a \ (\text{subst } s1 \ t1) \approx \text{Abst } b \ (\text{subst } s1 \ t2)$ 
  using subst-abst[of s1] by simp

from 5
have ii:  $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$ 
  using all-solutions-def by auto
from 5 i
have  $\text{nabla}1 \vdash a \# \text{subst } s1 \ t2$ 
  using Equ-elim( $\gamma$ ) by blast
with ii
have fresh:  $\forall (a,t) \in \text{set } ((a,t2) \# ys). \text{nabla}1 \vdash a \# \text{subst } s1 \ t$ 
  by simp

from i 5
have  $\text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ (\text{swap } [(a, b)] \ t2)$ 
  using 5 Equ-elim( $\gamma$ )[of  $\text{nabla}1 \ a \ \langle \text{subst } s1 \ t1 \rangle \ b \ \langle \text{subst } s1 \ t2 \rangle$ ]
  subst-swap-comm by simp
with eqsP1 have eqs:
   $\forall (t1,t2) \in \text{set } ((t1, \text{swap } [(a,b)] \ t2) \# xs). \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
  by auto

from subst fresh eqs
show ?case using all-solutions-def by simp
next
case (6 a xs ys)
then have subst:  $\text{nabla}1 \models \text{subst } (s1 \cdot []) \approx \text{subst } s1$ 
  using equ-refl subst-equ-def by simp

from 6
have fresh:  $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$ 
  using all-solutions-def by auto

from 6

```

```

have eqs:  $\forall (t1,t2) \in \text{set } xs. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
  using all-solutions-def by auto

from subst fresh eqs
show ?case using all-solutions-def by simp
next
case ( $\exists pi1 \ X \ pi2 \ xs \ ys$ )
then have subst:  $\text{nabla}1 \models \text{subst } (s1 \cdot []) \approx \text{subst } s1$ 
  using equ-refl subst-equ-def by simp

from  $\exists$ 
have i:  $\forall (t1,t2) \in \text{set } ((\text{Susp } pi1 \ X, \text{Susp } pi2 \ X) \# xs). \text{nabla}1 \vdash \text{subst } s1 \ t1$ 
 $\approx \text{subst } s1 \ t2$ 
   $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$ 
  using all-solutions-def by simp+

from  $\exists(1)$ 
have  $\text{nabla}1 \vdash \text{swap } pi1 \ (\text{look-up } X \ s1) \approx \text{swap } pi2 \ (\text{look-up } X \ s1)$ 
  using all-solutions-def subst-susp by simp
hence  $\forall a \in \text{ds } pi1 \ pi2. \text{nabla}1 \vdash a \# (\text{look-up } X \ s1)$ 
  using equ-pi1-pi2-dec by simp
hence  $\forall a \in \text{set } (\text{ds-list } pi1 \ pi2). \text{nabla}1 \vdash a \# \text{subst } s1 \ (\text{Susp } [] \ X)$ 
  using subst-susp ds-list-equ-ds[of pi1 pi2] by simp
hence  $\forall (a,t) \in \text{set } (\text{map } (\lambda a. a \#? \text{Susp } [] \ X) \ (\text{ds-list } pi1 \ pi2)). \text{nabla}1 \vdash a \#$ 
 $\text{subst } s1 \ t$ 
  by (auto split: prod.splits)
with  $\exists i(2)$ 
have fresh:
   $\forall (a,t) \in \text{set } (\text{map } (\lambda a. (a, \text{Susp } [] \ X)) \ (\text{ds-list } pi1 \ pi2) \ @ \ ys). \text{nabla}1 \vdash a \#$ 
 $\text{subst } s1 \ t$ 
  by auto

from  $i(1)$ 
have eqs:
   $\forall (t1,t2) \in \text{set } xs. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$ 
  by simp

from subst fresh eqs
show ?case using all-solutions-def by simp
next
case ( $\exists X \ t' \ pi \ xs \ ys$ )
hence  $\text{nabla}1 \vdash \text{subst } s1 \ (\text{Susp } pi \ X) \approx \text{subst } s1 \ t'$ 
  using all-solutions-def[of «((Susp pi X, t') # xs, ys)»] by auto
hence subst:  $\text{nabla}1 \models \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } pi) \ t')]) \approx \text{subst } s1$ 
  using unif-1 by simp

from  $\exists$ 
have eqs-old:

```

$\forall (t1,t2) \in \text{set } xs. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**and fresh-old:**  
 $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$   
**using all-solutions-def by auto**

**from eqs-old**  
**have eqs:**  $\forall (t1,t2) \in \text{set } xs.$   
 $\text{nabla}1 \vdash \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t1 \approx \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t2$   
**using unif-2a[OF subst] by auto**

**from fresh-old**  
**have fresh:**  $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t$   
**using unif-2b[OF subst] by auto**

**from eqs fresh**  
**have**  $(\text{nabla}1, (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')])) \in U \ (xs, ys)$   
**using all-solutions-def by simp**  
**hence**  $U: (\text{nabla}1, s1) \in U \ (\text{apply-subst } [(X, \text{swap } (\text{rev } \text{pi}) \ t')] \ (xs, ys))$   
**using problem-subst-comm by simp**

**from subst U**  
**show ?case by simp**

**next**  
**case**  $(9 \ X \ t' \ \text{pi} \ xs \ ys)$   
**hence**  $\text{nabla}1 \vdash \text{subst } s1 \ t' \approx \text{subst } s1 \ (\text{Susp } \text{pi} \ X)$   
**using all-solutions-def[of  $\langle (t', \text{Susp } \text{pi} \ X) \# xs, ys \rangle$ ] by simp**  
**hence subst:**  $\text{nabla}1 \models \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \approx \text{subst } s1$   
**using unif-1[OF equ-symm] by blast**

**from 9**  
**have eqs-old:**  
 $\forall (t1,t2) \in \text{set } xs. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**and fresh-old:**  
 $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$   
**using all-solutions-def by auto**

**from eqs-old**  
**have eqs:**  $\forall (t1,t2) \in \text{set } xs.$   
 $\text{nabla}1 \vdash \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t1 \approx \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t2$   
**using unif-2a[OF subst] by auto**

**from fresh-old**  
**have fresh:**  $\forall (a,t) \in \text{set } ys. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')]) \ t$   
**using unif-2b[OF subst] by auto**

**from eqs fresh**  
**have**  $(\text{nabla}1, (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) \ t')])) \in U \ (xs, ys)$

```

    using all-solutions-def by simp
  hence U: (nabla1, s1) ∈ U (apply-subst [(X, swap (rev pi) t')] (xs, ys))
    using problem-subst-comm by simp

  from subst U
  show ?case by simp
qed

Auxiliary lemma for completeness

lemma P1-from-P2-sred:
  assumes (nabla1, s1) ∈ U P2 and P1 ⊢ s2 ∼ P2
  shows (nabla1, s1 · s2) ∈ U P1
  using assms
proof(induction arbitrary: nabla1 s1 rule: s-red.induct[OF ⟨P1 ⊢ s2 ∼ P2⟩])
  case (1 xs ys)
  hence ∀ (t1, t2) ∈ set xs. nabla1 ⊢ subst (s1 · []) t1 ≈ subst (s1 · []) t2
    and fresh: ∀ (a, t) ∈ set ys. nabla1 ⊢ a ‡ subst (s1 · []) t
    using all-solutions-def by simp-all
  hence eqs:
    ∀ (t1, t2) ∈ set ((Unit, Unit) # xs). nabla1 ⊢ subst (s1 · []) t1 ≈ subst (s1 ·
[] ) t2
    using subst-unit[of ⟨(s1 · [])⟩] equ-refl[of nabla1 Unit] by auto
  from fresh eqs show ?case
    using all-solutions-def by simp
next
  case (2 t1 t2 t3 t4 xs ys)
  hence eqs-old: ∀ (t1, t2) ∈ set ((t1, t3) # (t2, t4) # xs). nabla1 ⊢ subst (s1 ·
[] ) t1 ≈ subst (s1 · [] ) t2
    and fresh: ∀ (a, t) ∈ set ys. nabla1 ⊢ a ‡ subst (s1 · []) t
    using all-solutions-def by simp-all
  hence nabla1 ⊢ subst (s1 · []) t1 ≈ subst (s1 · []) t3
    nabla1 ⊢ subst (s1 · []) t2 ≈ subst (s1 · []) t4 by simp-all
  hence nabla1 ⊢ subst (s1 · []) (Paar t1 t2) ≈ subst (s1 · []) (Paar t3 t4)
    using equ-paar by simp
  with eqs-old have eqs:
    ∀ (t1, t2) ∈ set ((Paar t1 t2, Paar t3 t4) # xs). nabla1 ⊢ subst (s1 · []) t1 ≈
subst (s1 · [] ) t2
    by auto
  from fresh eqs show ?case
    using all-solutions-def by simp
next
  case (3 f t1 t2 xs ys)
  hence eqs-old: ∀ (t1, t2) ∈ set ((t1, t2) # xs). nabla1 ⊢ subst (s1 · []) t1 ≈
subst (s1 · [] ) t2
    and fresh: ∀ (a, t) ∈ set ys. nabla1 ⊢ a ‡ subst (s1 · []) t
    using all-solutions-def by simp-all
  hence nabla1 ⊢ subst (s1 · []) (Func f t1) ≈ subst (s1 · []) (Func f t2)
    using equ-func by simp
  with eqs-old have eqs:

```

```

     $\forall (t1,t2) \in \text{set } ((\text{Func } f \ t1, \text{Func } f \ t2) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx$ 
 $\text{subst } (s1 \cdot []) \ t2$ 
    by auto
    from fresh eqs show ?case
    using all-solutions-def by simp
next
case (4 a t1 t2 xs ys)
hence eqs-old:  $\forall (t1,t2) \in \text{set } ((t1, t2) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx$ 
 $\text{subst } (s1 \cdot []) \ t2$ 
and fresh:  $\forall (a,t) \in \text{set } \text{ys}. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) \ t$ 
using all-solutions-def by simp-all
hence  $\text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ (\text{Abst } a \ t1) \approx \text{subst } (s1 \cdot []) \ (\text{Abst } a \ t2)$ 
using equ-abst-aa by simp
with eqs-old have eqs:
 $\forall (t1,t2) \in \text{set } ((\text{Abst } a \ t1, \text{Abst } a \ t2) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx$ 
 $\text{subst } (s1 \cdot []) \ t2$ 
by auto
from fresh eqs show ?case
using all-solutions-def by simp
next
case (5 a b t1 t2 xs ys)
hence eqs-old:  $\forall (t1,t2) \in \text{set } ((t1, \text{swap } [(a, b)] \ t2) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1$ 
 $\cdot []) \ t1 \approx \text{subst } (s1 \cdot []) \ t2$ 
and fresh:  $\forall (a,t) \in \text{set } ((a, t2) \# \text{ys}). \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) \ t$ 
using all-solutions-def by simp-all
hence  $\text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx \text{swap } [(a, b)] \ (\text{subst } (s1 \cdot []) \ t2)$ 
 $\text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) \ t2$ 
using subst-swap-comm[of  $\langle (s1 \cdot []) \rangle \langle [(a, b)] \rangle \ t2]$  by simp-all
hence  $\text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ (\text{Abst } a \ t1) \approx \text{subst } (s1 \cdot []) \ (\text{Abst } b \ t2)$ 
using equ-abst-ab[OF 5(1)  $\langle \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) \ t2 \rangle$ ] subst-abst by simp
with eqs-old have eqs:
 $\forall (t1,t2) \in \text{set } ((\text{Abst } a \ t1, \text{Abst } b \ t2) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx$ 
 $\text{subst } (s1 \cdot []) \ t2$ 
by auto
from fresh eqs show ?case
using all-solutions-def by simp
next
case (6 a xs ys)
hence  $\forall (t1,t2) \in \text{set } \text{xs}. \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx \text{subst } (s1 \cdot []) \ t2$ 
and fresh:  $\forall (a,t) \in \text{set } \text{ys}. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) \ t$ 
using all-solutions-def by simp-all
hence eqs:
 $\forall (t1,t2) \in \text{set } ((\text{Atom } a, \text{Atom } a) \# \text{xs}). \text{nabla}1 \vdash \text{subst } (s1 \cdot []) \ t1 \approx \text{subst}$ 
 $(s1 \cdot []) \ t2$ 
using subst-unit[of  $\langle (s1 \cdot []) \rangle$ ] equ-refl[of  $\text{nabla}1 \ \text{Unit}$ ] by auto
from fresh eqs show ?case
using all-solutions-def by simp
next
case (7 pi1 X pi2 xs ys)

```

**hence** *eqs-old*:  $\forall (t1, t2) \in \text{set } xs. \text{nabla}1 \vdash \text{subst } (s1 \cdot []) t1 \approx \text{subst } (s1 \cdot []) t2$   
**and** *fresh-old*:  
 $\forall (a, t) \in \text{set } (\text{map } (\lambda a. (a, \text{Susp } [] X)) (\text{ds-list } \text{pi1 } \text{pi2}) @ \text{ys}). \text{nabla}1 \vdash a \#$   
 $\text{subst } (s1 \cdot []) t$   
**using** *all-solutions-def* **by** *simp-all*  
**hence**  $\forall c \in \text{set } (\text{ds-list } \text{pi1 } \text{pi2}). \text{nabla}1 \vdash c \# \text{subst } (s1 \cdot []) (\text{Susp } [] X)$   
**by** (*auto split: prod.splits*)  
**hence**  $\forall c \in \text{ds } \text{pi1 } \text{pi2}. \text{nabla}1 \vdash c \# \text{subst } (s1 \cdot []) (\text{Susp } [] X)$   
**using** *ds-list-equ-ds* **by** *simp*  
**hence**  $\forall c \in \text{ds } \text{pi1 } \text{pi2}. \text{nabla}1 \vdash c \# (\text{look-up } X (s1 \cdot []))$   
**using** *subst-susp*[of  $\langle (s1 \cdot []) \rangle \langle [] \rangle X$ ] *swap-id* **by** *simp*  
**hence**  $\text{nabla}1 \vdash \text{subst } (s1 \cdot []) (\text{Susp } \text{pi1 } X) \approx \text{subst } (s1 \cdot []) (\text{Susp } \text{pi2 } X)$   
**using** *equ-equiv-pi* *subst-susp*[of  $\langle (s1 \cdot []) \rangle - X$ ] **by** *auto*  
**with** *eqs-old* *fresh-old* **have**  
*eqs*:  $\forall (t1, t2) \in \text{set } ((\text{Susp } \text{pi1 } X, \text{Susp } \text{pi2 } X) \# xs).$   
 $\text{nabla}1 \vdash \text{subst } (s1 \cdot []) t1 \approx \text{subst } (s1 \cdot []) t2$   
**and** *fresh*:  $\forall (a, t) \in \text{set } \text{ys}. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot []) t$   
**by** *simp-all*  
**then show** *?case*  
**using** *all-solutions-def* **by** *simp*  
**next**  
**case** (*8 X t' pi xs ys*)  
**hence**  $(\text{nabla}1, s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) \in U (xs, \text{ys})$   
**using** *problem-subst-comm* **by** *simp*  
**hence** *eqs-old*:  
 $\forall (t1, t2) \in \text{set } xs.$   
 $\text{nabla}1 \vdash \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) t1 \approx \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi})$   
 $t')]) t2$   
**and** *fresh*:  $\forall (a, t) \in \text{set } \text{ys}. \text{nabla}1 \vdash a \# \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) t$   
**using** *all-solutions-def* **by** *simp-all*  
  
**have**  $\text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) (\text{Susp } \text{pi } X) = \text{subst } s1 (\text{swap } \text{pi } (\text{swap}$   
 $(\text{rev } \text{pi}) t'))$   
**using** *subst-susp* *subst-swap-comm* *subst-comp-expand* *eq-fst-iff* *look-up.simps(2)*  
*snd-eqD* **by** *metis*  
**also have**  $\dots = \text{swap } \text{pi } (\text{swap } (\text{rev } \text{pi}) (\text{subst } s1 t'))$   
**using** *subst-swap-comm* **by** *simp*  
**hence**  $\text{nabla}1 \vdash \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) (\text{Susp } \text{pi } X) \approx \text{subst } s1 t'$   
**using** *equ-refl* *calculation* *rev-pi-pi-equ*[of  $\text{nabla}1 \text{ pi } \langle \text{subst } s1 t' \rangle$ ]  
 $\text{equ-trans}$ [of  $\text{nabla}1 \langle \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) (\text{Susp } \text{pi } X) \rangle$   
 $\langle \text{swap } (\text{rev } \text{pi}) (\text{swap } \text{pi } (\text{subst } s1 t')) \rangle \langle \text{subst } s1 t' \rangle$ ]  
*swap-inv-side* **by** *auto*  
  
**with** *8 eqs-old*  
**have** *eqs*:  $\forall (t1, t2) \in \text{set } ((\text{Susp } \text{pi } X, t') \# xs).$   
 $\text{nabla}1 \vdash \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi}) t')]) t1 \approx \text{subst } (s1 \cdot [(X, \text{swap } (\text{rev } \text{pi})$   
 $t')]) t2$   
**using** *subst-comp-expand* *subst-not-occurs* **by** *simp*

**from** *fresh eqs show ?case*  
**using** *all-solutions-def by simp*  
**next**  
**case** (*9 X t' pi xs ys*)  
**hence** (*nabla1, s1 · [(X, swap (rev pi) t')] ∈ U (xs, ys)*)  
**using** *problem-subst-comm by simp*  
**hence** *eqs-old:*  
 $\forall (t1, t2) \in \text{set } xs.$   
*nabla1 ⊢ subst (s1 · [(X, swap (rev pi) t')]) t1 ≈ subst (s1 · [(X, swap (rev pi) t')]) t2*  
**and** *fresh:*  $\forall (a, t) \in \text{set } ys. \text{nabla1} \vdash a \# \text{subst} (s1 \cdot [(X, \text{swap} (\text{rev } \text{pi}) t')]) t$   
**using** *all-solutions-def by simp-all*

**have** *subst (s1 · [(X, swap (rev pi) t')]) (Susp pi X) = subst s1 (swap pi (swap (rev pi) t'))*  
**using** *subst-susp subst-swap-comm subst-comp-expand eq-fst-iff look-up.simps(2) snd-eqD by metis*  
**also have** *... = swap pi (swap (rev pi) (subst s1 t'))*  
**using** *subst-swap-comm by simp*  
**hence** *nabla1 ⊢ subst (s1 · [(X, swap (rev pi) t')]) (Susp pi X) ≈ subst s1 t'*  
**using** *equ-refl calculation rev-pi-pi-equ[of nabla1 pi ⟨subst s1 t'⟩]*  
 $\text{equ-trans}[\text{of } \text{nabla1} \langle \text{subst} (s1 \cdot [(X, \text{swap} (\text{rev } \text{pi}) t')]) (\text{Susp } \text{pi } X) \rangle$   
 $\langle \text{swap} (\text{rev } \text{pi}) (\text{swap } \text{pi} (\text{subst } s1 t')) \rangle \langle \text{subst } s1 t' \rangle]$   
*swap-inv-side by auto*  
**hence** *nabla1 ⊢ subst s1 t' ≈ subst (s1 · [(X, swap (rev pi) t')]) (Susp pi X)*  
**using** *equ-symm by simp*

**with** *9 eqs-old*  
**have** *eqs:*  $\forall (t1, t2) \in \text{set} ((t', \text{Susp } \text{pi } X) \# xs).$   
*nabla1 ⊢ subst (s1 · [(X, swap (rev pi) t')]) t1 ≈ subst (s1 · [(X, swap (rev pi) t')]) t2*  
**using** *subst-comp-expand subst-not-occurs by simp*

**from** *fresh eqs show ?case*  
**using** *all-solutions-def by simp*  
**qed**

**lemma** *P1-to-P2-cred:*  
**assumes** (*nabla1, s1*) ∈ *U P1*  
**and** *P1 ⊢ nabla2 → P2*  
**shows** (*nabla1, s1*) ∈ *U P2* ∧ (*nabla1* ≡ (*subst s1*) *nabla2*)  
**using** *assms*  
**proof** (*induction arbitrary: nabla1 s1 rule: c-red.induct[OF ⟨P1 ⊢ nabla2 → P2⟩]*)  
**case** (*2 a t1 t2 ys*)  
**hence** *fresh-old:*  $\forall (a, t) \in \text{set} ((a, \text{Paar } t1 t2) \# ys). \text{nabla1} \vdash a \# \text{subst } s1 t$   
**and** *eqs:*  $\forall (t1, t2) \in \text{set} []. \text{nabla1} \vdash \text{subst } s1 t1 \approx \text{subst } s1 t2$   
**using** *all-solutions-def by simp-all*  
**hence** *nabla1 ⊢ a # subst s1 (Paar t1 t2)*  
**by** *simp*

**hence**  $nabla1 \vdash a \# \text{subst } s1 \ t1 \ nabla1 \vdash a \# \text{subst } s1 \ t2$   
**using** *Fresh-elim*(5)[of  $nabla1 \ a \ \langle \text{subst } s1 \ t1 \rangle$   
 $\langle \text{subst } s1 \ t2 \rangle$ ] **by** *auto*  
**with** *fresh-old* **have**  
*fresh*:  $\forall (a,t) \in \text{set } ((a, t1) \# (a, t2) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t$   
**by** *auto*  
**from** *fresh eqs*  
**show** *?case*  
**using** *all-solutions-def ext-subst-def* **by** *simp*  
**next**  
**case** ( $\exists \ a \ f \ t \ \text{ys}$ )  
**hence** *fresh-old*:  $\forall (a,t) \in \text{set } ((a, \text{Func } f \ t) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t$   
**and** *eqs*:  $\forall (t1,t2) \in \text{set } []. \ nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *all-solutions-def* **by** *simp-all*  
**hence**  $nabla1 \vdash a \# \text{subst } s1 \ (\text{Func } f \ t)$   
**by** *simp*  
**hence**  $nabla1 \vdash a \# \text{subst } s1 \ t$   
**using** *Fresh-elim*(6) **by** *auto*  
**with** *fresh-old* **have**  
*fresh*:  $\forall (a,t) \in \text{set } ((a, t) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t$   
**by** *auto*  
**from** *fresh eqs*  
**show** *?case*  
**using** *all-solutions-def ext-subst-def* **by** *simp*  
**next**  
**case** ( $\exists \ a \ b \ t \ \text{ys}$ )  
**hence** *fresh-old*:  $\forall (a,t') \in \text{set } ((a, \text{Abst } b \ t) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t'$   
**and** *eqs*:  $\forall (t1,t2) \in \text{set } []. \ nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *all-solutions-def* **by** *simp-all*  
**hence**  $nabla1 \vdash a \# \text{subst } s1 \ (\text{Abst } b \ t)$   
**by** *simp*  
**hence**  $nabla1 \vdash a \# \text{subst } s1 \ t$   
**using** *5 Fresh-elim*(1)[of  $nabla1 \ a \ b \ \langle \text{subst } s1 \ t \rangle$ ] **by** *auto*  
**with** *fresh-old* **have**  
*fresh*:  $\forall (a,t') \in \text{set } ((a, t) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t'$   
**by** *auto*  
**from** *fresh eqs*  
**show** *?case*  
**using** *all-solutions-def ext-subst-def* **by** *simp*  
**next**  
**case** ( $\exists \ b \ \text{pi } X \ \text{ys}$ )  
**hence** *fresh-old*:  $\forall (a,t) \in \text{set } ((b, \text{Susp } \text{pi } X) \# \text{ys}). \ nabla1 \vdash a \# \text{subst } s1 \ t$   
**and** *eqs*:  $\forall (t1,t2) \in \text{set } []. \ nabla1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *all-solutions-def* **by** *simp-all*  
**hence** *fresh*:  $\forall (a,t) \in \text{set } \text{ys}. \ nabla1 \vdash a \# \text{subst } s1 \ t$  **by** *simp*  
**from** *fresh-old* **have**  
 $nabla1 \vdash b \# \text{subst } s1 \ (\text{Susp } \text{pi } X)$  **by** *simp*  
**hence**  $nabla1 \vdash \text{swapas } (\text{rev } \text{pi}) \ b \# \text{subst } s1 \ (\text{Susp } [] \ X)$   
**using** *fresh-swap-left subst-susp* **by** *simp*

**hence**  $ext: nabra1 \models (subst\ s1)\ \{(swapas\ (rev\ pi)\ b,\ X)\}$   
**using**  $ext\text{-subst}\text{-def}$  **by**  $simp$   
**from**  $fresh\ eqs\ ext$   
**show**  $?case$  **using**  $all\text{-solutions}\text{-def}$  **by**  $simp$   
**qed** ( $auto\ simp\ add: all\text{-solutions}\text{-def}\ ext\text{-subst}\text{-def}$ )

**lemma**  $P1\text{-from}\text{-}P2\text{-cred}$ :

**assumes**  $(nabra1, s1) \in U\ P2$   
**and**  $P1 \vdash nabra2 \rightarrow P2$  **and**  $nabra3 \models (subst\ s1)\ nabra2$   
**shows**  $(nabra1 \cup nabra3, s1) \in U\ P1$   
**using**  $assms$

**proof**( $induction\ arbitrary: nabra1\ s1\ rule: c\text{-red.}\ induct[OF\ \langle P1 \vdash nabra2 \rightarrow P2 \rangle]$ )

**case**  $(2\ a\ t1\ t2\ xs)$

**hence**  $fresh\text{-old}$ :

$\forall (b, t) \in set\ ((a, t1) \# (a, t2) \# xs). nabra1 \vdash b \# subst\ s1\ t$   
**and**  $\forall (t1, t2) \in set\ []. nabra1 \vdash subst\ s1\ t1 \approx subst\ s1\ t2$   
**using**  $all\text{-solutions}\text{-def}$  **by**  $auto$

**hence**  $weak1: \forall (b, t) \in set\ xs. (nabra1 \cup nabra3) \vdash b \# subst\ s1\ t$

$\forall (t1, t2) \in set\ []. (nabra1 \cup nabra3) \vdash subst\ s1\ t1 \approx subst\ s1\ t2$

**using**  $fresh\text{-weak}$  **by**  $auto$

**from**  $fresh\text{-old}$  **have**  $nabra1 \vdash a \# subst\ s1\ t1\ nabra1 \vdash a \# subst\ s1\ t2$

**by**  $simp+$

**hence**  $nabra1 \vdash a \# subst\ s1\ (Paar\ t1\ t2)$

**using**  $fresh\text{-paar}\ subst\text{-paar}$  **by**  $auto$

**hence**  $(nabra1 \cup nabra3) \vdash a \# subst\ s1\ (Paar\ t1\ t2)$

**using**  $fresh\text{-weak}$  **by**  $auto$

**with**  $weak1$

**have**  $fresh: \forall (b, t) \in set\ ((a, Paar\ t1\ t2) \# xs). (nabra1 \cup nabra3) \vdash b \# subst\ s1\ t$

**by**  $simp$

**with**  $weak1(2)$  **show**  $?case$

**using**  $all\text{-solutions}\text{-def}\ ext\text{-subst}\text{-def}$  **by**  $simp$

**next**

**case**  $(3\ a\ f\ t'\ xs)$

**hence**  $fresh\text{-old}$ :

$\forall (b, t) \in set\ ((a, t') \# xs). nabra1 \vdash b \# subst\ s1\ t$   
**and**  $\forall (t1, t2) \in set\ []. nabra1 \vdash subst\ s1\ t1 \approx subst\ s1\ t2$   
**using**  $all\text{-solutions}\text{-def}$  **by**  $auto$

**hence**  $weak1: \forall (b, t) \in set\ xs. (nabra1 \cup nabra3) \vdash b \# subst\ s1\ t$

$\forall (t1, t2) \in set\ []. (nabra1 \cup nabra3) \vdash subst\ s1\ t1 \approx subst\ s1\ t2$

**using**  $fresh\text{-weak}$  **by**  $auto$

**from**  $fresh\text{-old}$  **have**  $nabra1 \vdash a \# subst\ s1\ t'$

**by**  $simp$

**hence**  $nabra1 \vdash a \# subst\ s1\ (Func\ f\ t')$

**using**  $fresh\text{-paar}\ subst\text{-paar}$  **by**  $auto$

**hence**  $(nabra1 \cup nabra3) \vdash a \# subst\ s1\ (Func\ f\ t')$

**using**  $fresh\text{-weak}$  **by**  $auto$

**with**  $weak1$

**have**  $fresh: \forall (b, t) \in set\ ((a, Func\ f\ t') \# xs). (nabra1 \cup nabra3) \vdash b \# subst\ s1\ t$

**by**  $simp$

**with** *weak1*(2) **show** ?*case*  
**using** *all-solutions-def ext-subst-def* **by** *simp*  
**next**  
**case** (5 *a b t' xs*)  
**hence** *fresh-old*:  
 $\forall (c,t) \in \text{set } ((a,t')\#xs). \text{nabla}1 \vdash c \# \text{subst } s1 \ t$   
**and**  $\forall (t1, t2) \in \text{set } []. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *all-solutions-def* **by** *auto*  
**hence** *weak1*:  $\forall (c,t) \in \text{set } xs. (\text{nabla}1 \cup \text{nabla}3) \vdash c \# \text{subst } s1 \ t$   
 $\forall (t1, t2) \in \text{set } []. (\text{nabla}1 \cup \text{nabla}3) \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *fresh-weak* **by** *auto*  
**from** *fresh-old* **have**  $\text{nabla}1 \vdash a \# \text{subst } s1 \ t'$   
**by** *simp*  
**hence**  $\text{nabla}1 \vdash a \# \text{subst } s1 \ (\text{Abst } b \ t')$   
**using** *fresh-abst-ab[OF <nabla1 \vdash a \# subst s1 t'> 5(1)] subst-abst* **by** *auto*  
**hence**  $(\text{nabla}1 \cup \text{nabla}3) \vdash a \# \text{subst } s1 \ (\text{Abst } b \ t')$   
**using** *fresh-weak* **by** *auto*  
**with** *weak1*  
**have** *fresh*:  $\forall (b,t) \in \text{set } ((a, \text{Abst } b \ t')\#xs). (\text{nabla}1 \cup \text{nabla}3) \vdash b \# \text{subst } s1 \ t$   
**by** *simp*  
**with** *weak1*(2) **show** ?*case*  
**using** *all-solutions-def ext-subst-def* **by** *simp*  
**next**  
**case** (7 *b pi X xs*)  
**hence** *fresh-old*:  $\forall (a,t) \in \text{set } xs. \text{nabla}1 \vdash a \# \text{subst } s1 \ t$   
**and** *eqs*:  $\forall (t1,t2) \in \text{set } []. \text{nabla}1 \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *all-solutions-def* **by** *simp-all*  
**hence** *weak1*:  $\forall (a,t) \in \text{set } xs. (\text{nabla}1 \cup \text{nabla}3) \vdash a \# \text{subst } s1 \ t$   
 $\forall (t1, t2) \in \text{set } []. (\text{nabla}1 \cup \text{nabla}3) \vdash \text{subst } s1 \ t1 \approx \text{subst } s1 \ t2$   
**using** *fresh-weak* **by** *auto*  
**from** 7 **have**  $\text{nabla}3 \vdash \text{swapas } (\text{rev } pi) \ b \# \text{subst } s1 \ (\text{Susp } [] \ X)$   
**using** *ext-subst-def* **by** *auto*  
**hence**  $\text{nabla}3 \vdash b \# \text{subst } s1 \ (\text{Susp } pi \ X)$   
**using** *fresh-swap-right subst-susp* **by** *auto*  
**hence** *weak2*:  $(\text{nabla}1 \cup \text{nabla}3) \vdash b \# \text{subst } s1 \ (\text{Susp } pi \ X)$   
**using** *fresh-weak[of nabla3 b - nabla1] Un-commute[of nabla3 nabla1]* **by** *auto*  
**with** *weak1*(1)  
**have**  $\forall (a,t) \in \text{set } ((b, \text{Susp } pi \ X)\#xs). (\text{nabla}1 \cup \text{nabla}3) \vdash a \# \text{subst } s1 \ t$   
**by** *simp*  
**with** *weak2* **show** ?*case*  
**using** *ext-subst-def all-solutions-def* **by** *simp*  
**qed** (*auto simp add: ext-subst-def all-solutions-def fresh-weak*)

**lemma** *P1-to-P2-red-plus1*:  
**assumes**  $P1 \models (\text{nabla}, s) \Rightarrow P2$   
**and**  $(\text{nabla}1, s1) \in U \ P1$   
**shows**  $(\text{nabla}1, s1) \in U \ P2$   
**using** *assms*  
**proof** (*induction P1 \equiv P1 nabla \equiv (nabla, s) P2 \equiv P2 arbitrary: s nabla1 nabla s1 P1*)

*P2 rule: red-plus.induct*  
**case** (*sred-single*  $P1\ s\ P2\ nabla1\ s1$ )  
**have**  $P1 \vdash s \rightsquigarrow P2\ (nabla1, s1) \in U\ P1$  **by** *fact+*  
**then show**  $(nabla1, s1) \in U\ P2$   
**using** *P1-to-P2-sred* **by** *blast*  
**next**  
**case** (*cred-single*  $P1\ nabla1\ P2\ nabla2\ s1$ )  
**have**  $P1 \vdash nabla1 \rightarrow P2\ (nabla2, s1) \in U\ P1$  **by** *fact+*  
**then show**  $(nabla2, s1) \in U\ P2$   
**using** *P1-to-P2-cred* **by** *blast*  
**next**  
**case** (*sred-step*  $P1\ s\ P2\ nabla2\ s2\ P3\ nabla1\ s1$ )  
**have**  $P1 \vdash s \rightsquigarrow P2$  **and**  $(nabla1, s1) \in U\ P1$  **by** *fact+*  
**then have**  $(nabla1, s1) \in U\ P2$  **using** *P1-to-P2-sred* **by** *blast*  
**moreover**  
**have** *IH*:  $(nabla1, s1) \in U\ P2 \implies (nabla1, s1) \in U\ P3$  **by** *fact*  
**ultimately**  
**show**  $(nabla1, s1) \in U\ P3$  **by** *simp*  
**next**  
**case** (*cred-step*  $P1\ nabla1\ P2\ nabla2\ P3\ nabla3\ s1$ )  
**have**  $P1 \vdash nabla1 \rightarrow P2$  **and**  $(nabla3, s1) \in U\ P1$  **by** *fact+*  
**then have**  $(nabla3, s1) \in U\ P2$  **using** *P1-to-P2-cred* **by** *blast*  
**moreover**  
**have** *IH*:  $(nabla3, s1) \in U\ P2 \implies (nabla3, s1) \in U\ P3$  **by** *fact*  
**ultimately show**  $(nabla3, s1) \in U\ P3$  **by** *simp*  
**qed**

**lemma** *P1-to-P2-red-plus3*:  
**assumes**  $P1 \models (nabla, s) \Rightarrow P2$   
**and**  $(nabla1, s1) \in U\ P1$   
**shows**  $nabla1 \models (subst\ s1)\ nabla$   
**using** *assms*  
**proof** (*induction*  $P1 \equiv P1\ nabla \equiv (nabla, s)\ P2 \equiv P2$  *arbitrary: s nabla1 nabla s1 P1*)  
*P2 rule: red-plus.induct*  
**case** (*sred-single*  $P1\ s\ P2\ nabla1\ s1$ )  
**then show**  $nabla1 \models (subst\ s1)\ \{\}$  **by** (*simp add: ext-subst-def*)  
**next**  
**case** (*cred-single*  $P1\ nabla1\ P2\ nabla2\ s1$ )  
**have**  $P1 \vdash nabla1 \rightarrow P2\ (nabla2, s1) \in U\ P1$  **by** *fact+*  
**then show**  $nabla2 \models (subst\ s1)\ nabla1$  **using** *P1-to-P2-cred* **by** *blast*  
**next**  
**case** (*sred-step*  $P1\ s\ P2\ nabla2\ s2\ P3\ nabla1\ s1$ )  
**have**  $P1 \vdash s \rightsquigarrow P2\ (nabla1, s1) \in U\ P1$  **by** *fact+*  
**then have**  $(nabla1, s1) \in U\ P2$  **using** *P1-to-P2-sred* **by** *blast*  
**moreover have** *IH*:  $(nabla1, s1) \in U\ P2 \implies nabla1 \models (subst\ s1)\ nabla2$  **by**  
*fact*  
**ultimately show**  $nabla1 \models (subst\ s1)\ nabla2$  **by** *simp*  
**next**  
**case** (*cred-step*  $P1\ nabla1\ P2\ nabla2\ P3\ nabla3\ s1$ )

**have**  $IH: (nabla3, s1) \in U P2 \implies \nabla3 \models (subst\ s1)\ \nabla2$  **by** *fact*  
**have**  $as: P1 \vdash \nabla1 \rightarrow P2 (\nabla3, s1) \in U P1$  **by** *fact+*

**from**  $as$  **have**  $\nabla3 \models (subst\ s1)\ \nabla1$  **using** *P1-to-P2-cred* **by** *blast*  
**moreover**  
**from**  $as$  **have**  $(\nabla3, s1) \in U P2$  **using** *P1-to-P2-cred* **by** *blast*  
**then** **have**  $\nabla3 \models (subst\ s1)\ \nabla2$  **using**  $IH$  **by** *blast*  
**ultimately** **show**  $\nabla3 \models (subst\ s1)\ (\nabla2 \cup \nabla1)$   
**by**(*auto simp add: ext-subst-def*)

**qed**

**lemma** *P1-to-P2-red-plus2*:  
**assumes**  $P1 \models (\nabla, s) \Rightarrow P2$   
**and**  $(\nabla1, s1) \in U P1$   
**shows**  $\nabla1 \models subst\ (s1 \cdot s) \approx subst\ s1$   
**using** *assms*  
**proof**(*induction P1 $\equiv$ P1  $\nabla$  $\equiv$ ( $\nabla, s$ ) P2 $\equiv$ P2 arbitrary: s  $\nabla1$   $\nabla$  s1 P1*  
*P2 rule: red-plus.induct*)  
**case** (*sred-single P1 s P2  $\nabla1$  s1*)  
**have**  $P1 \vdash_s \rightsquigarrow P2 (\nabla1, s1) \in U P1$  **by** *fact+*  
**then** **show**  $\nabla1 \models subst\ (s1 \cdot s) \approx_{subst\ s1}$   
**using** *P1-to-P2-sred* **by** *blast*

**next**  
**case** (*cred-single P1  $\nabla1$  P2  $\nabla2$  s1*)  
**show**  $\nabla2 \models subst\ (s1 \cdot []) \approx_{subst\ s1}$   
**by** (*simp add: subst-equ-refl*)

**next**  
**case** (*sred-step P1 s P2  $\nabla2$  s2 P3  $\nabla1$  s1*)  
**have**  $IH: (\nabla1, s1) \in U P2 \implies \nabla1 \models subst\ (s1 \cdot s2) \approx_{subst\ s1}$  **by** *fact*  
**have**  $as: P1 \vdash_s \rightsquigarrow P2 (\nabla1, s1) \in U P1$  **by** *fact+*

**from**  $as$  **have**  $(\nabla1, s1) \in U P2$  **using** *P1-to-P2-sred* **by** *blast*  
**with**  $IH$  **have**  $\nabla1 \models subst\ (s1 \cdot s2) \approx_{subst\ s1}$  **by** *blast*  
**then** **have**  $\nabla1 \models subst\ ((s1 \cdot s2) \cdot s) \approx_{subst\ s1}$   
**by** (*simp add: subst-cancel-right*)  
**moreover**  
**from**  $as$  **have**  $\nabla1 \models subst\ (s1 \cdot s) \approx_{subst\ s1}$   
**using** *P1-to-P2-sred* **by** *blast*  
**ultimately**  
**show**  $\nabla1 \models subst\ (s1 \cdot (s2 \cdot s)) \approx_{subst\ s1}$   
**using** *subst-assoc subst-trans* **by** *metis*

**next**  
**case** (*cred-step P1  $\nabla1$  P2  $\nabla2$  P3*)  
**show**  $\nabla1 \models subst\ (s1 \cdot []) \approx_{subst\ s1}$   
**by** (*simp add: subst-equ-refl*)

**qed**

**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**  $(nabla1 \cup nabla3, (s1 \cdot s)) \in U P1$   
**using** *assms*  
**proof**(*induction*  $P1 \equiv P1$   $nabla \equiv (nabla, s)$   $P2 \equiv P2$  *arbitrary: s nabla1 nabla3 s1 P1 P2* *rule: red-plus.induct*)  
**case** (*sred-single*  $P1$   $s$   $P2$   $nabla1$   $nabla3$ )  
**have**  $P1 \vdash s \rightsquigarrow P2$   $(nabla1, s1) \in U P2$  **by** *fact+*  
**then have**  $(nabla1, s1 \cdot s) \in U P1$   
**using** *P1-from-P2-sred* **by** *simp*  
**then show**  $(nabla1 \cup nabla3, s1 \cdot s) \in U P1$   
**using** *P1-from-P2-sred solution-weak* **by** *simp*  
**next**  
**case** (*cred-single*  $P1$   $nabla$   $P2$   $nabla1$   $nabla3$ )  
**have**  $P1 \vdash nabla \rightarrow P2$   
 $(nabla1, s1) \in U P2$   
 $nabla3 \models (subst\ s1)\ nabla$  **by** *fact+*  
**hence**  $(nabla1 \cup nabla3, s1) \in U P1$   
**using** *P1-from-P2-cred* **by** *simp*  
**then show**  $(nabla1 \cup nabla3, s1 \cdot []) \in U P1$   
**using** *solution-comp-id* **by** *auto*  
**next**  
**case** (*sred-step*  $P1$   $s$   $P2$   $nabla2$   $s2$   $P3$   $nabla1$   $nabla3$ )  
**have** *IH*:  $(nabla1, s1) \in U P3 \implies$   
 $nabla3 \models (subst\ s1)\ nabla2 \implies (nabla1 \cup nabla3, s1 \cdot s2) \in U P2$  **by** *fact+*  
**have** *as*:  $(nabla1, s1) \in U P3$   
 $nabla3 \models (subst\ s1)\ nabla2$   
 $P1 \vdash s \rightsquigarrow P2$  **by** *fact+*  
**hence**  $(nabla1 \cup nabla3, s1 \cdot s2) \in U P2$   
**using** *IH* **by** *simp*  
**hence**  $(nabla1 \cup nabla3, (s1 \cdot s2) \cdot s) \in U P1$   
**using** *as(3)*  
**by** (*simp add: P1-from-P2-sred*)  
**then show**  $(nabla1 \cup nabla3, s1 \cdot (s2 \cdot s)) \in U P1$   
**using** *solutions-subst-assoc* **by** *simp*  
**next**  
**case** (*cred-step*  $P1$   $nabla$   $P2$   $nabla2$   $P3$ )  
**have** *IH*:  $(nabla1, s1) \in U P3 \implies$   
 $nabla3 \models (subst\ s1)\ nabla2 \implies (nabla1 \cup nabla3, s1 \cdot []) \in U P2$  **by** *fact*  
**have** *as*:  $P1 \vdash nabla \rightarrow P2$   
 $(nabla1, s1) \in U P3$   
 $nabla3 \models (subst\ s1)\ (nabla2 \cup nabla)$  **by** *fact+*  
**have** *ext-substs*:  $nabla3 \models (subst\ s1)\ (nabla2)$   
 $nabla3 \models (subst\ s1)\ nabla$   
**using** *ext-subst-strong[OF as(3)]* **by** *simp+*  
**hence** *a*:  $(nabla1 \cup nabla3, s1 \cdot []) \in U P2$   
**using** *IH as(2)* **by** *simp*  
**hence**  $(nabla1 \cup nabla3 \cup nabla3, s1 \cdot []) \in U P1$   
**using** *as(1) ext-substs(2) P1-from-P2-cred solution-comp-id* **by** *blast*  
**then show**  $(nabla1 \cup nabla3, s1 \cdot []) \in U P1$

**unfolding** *all-solutions-def* **using** *Un-absorb* **by** *simp*  
**qed**

**lemma** *mgu*:

**assumes**  $P \models (nabla, s) \Rightarrow (\[], \[])$   
**shows**  $mgu\ P\ (nabla, s) \wedge idem\ (nabla, s)$

**proof**(*rule mgu-idem*)

**have**  $i: (\{\}, \[]) \in U\ (\[], \[])$

**unfolding** *all-solutions-def* **by** *simp*

**then show**  $(nabla, s) \in U\ P$

**using** *P1-from-P2-red-plus*[*OF assms i*]

*ext-subst-id solution-comp-id*(2) **by** *simp*

**show**  $\forall (nabla_2, s_2) \in U\ P. \ nabla_2 \models (subst\ s_2)\ nabla \wedge$   
 $nabla_2 \models subst\ (s_2 \cdot s) \approx subst\ s_2$

**proof**

**fix**  $x$

**assume**  $x \in U\ P$

**then show** *case*  $x$  *of*  $(nabla_2, s_2) \Rightarrow$

$nabla_2 \models (subst\ s_2)\ nabla \wedge$

$nabla_2 \models subst\ (s_2 \cdot s) \approx subst\ s_2$

**proof** (*cases*  $x$ )

**case** (*Pair*  $nabla_2\ s_2$ )

**hence**  $(nabla_2, s_2) \in U\ P$

**using**  $\langle x \in U\ P \rangle$  **by** *auto*

**hence**  $nabla_2 \models (subst\ s_2)\ nabla \wedge nabla_2 \models subst\ (s_2 \cdot s) \approx subst\ s_2$

**using** *assms P1-to-P2-red-plus2 P1-to-P2-red-plus3* **by** *auto+*

**with** *Pair* **show** *?thesis* **by** *simp*

**qed**

**qed**

**qed**

## 11 Termination

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

**lemma** *apply-subst-equivalence*:

**shows**  $apply\ subst\ s\ P = (apply\ subst\ eprobs\ s\ (fst\ P), apply\ subst\ fprobs\ s\ (snd\ P))$

**unfolding** *apply-subst-def apply-subst-eprobs-def apply-subst-fprobs-def* **by** *simp*

**fun** *size-trm* :: *trm*  $\Rightarrow$  *nat*

**where**

*size-trm* (*Unit*) = 1 |

$size-trm (Atom a) = 1 \mid$   
 $size-trm (Susp pi X) = 1 \mid$   
 $size-trm (Abst a t) = 1 + size-trm t \mid$   
 $size-trm (Func F t) = 1 + size-trm t \mid$   
 $size-trm (Paar t t') = 1 + (size-trm t) + (size-trm t')$

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

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

**lemma** *size-swap* [*simp*]:  $size-trm (swap pi t) = size-trm t$   
**by** (*induct t*) *auto*

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

**lemma** *vars-term-finite* [*simp*]:  $finite (vars-trm t)$   
**by** (*induct t*) *auto*

**lemma** *vars-eprobs-finite* [*simp*]:  $finite (vars-eprobs P)$   
**by** (*induct P*) *auto*

**lemma** *not-occurs-trm*:  $\neg occurs X t \longrightarrow X \notin vars-trm t$   
**by** (*induct t*) *auto*

**lemma** *not-occurs-subst*:  $\neg occurs X t1 \longrightarrow X \notin vars-trm (subst [(X, swap pi2 t1)] t2)$   
**using** *subst-susp not-occurs-trm* **by** (*induct t2*) (*auto simp add: vars-swap*)

**lemma** *not-occurs-list*:  $\neg occurs X t \longrightarrow$   
 $X \notin vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs)$   
**using** *not-occurs-subst apply-subst-eprobs-def* **by** (*induct xs*) *auto*

**lemma** *vars-equ*:  
**assumes**  $\neg occurs X t1$  **and**  $occurs X t2$

**shows**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] t2) = (\text{vars-trm } t1 \cup \text{vars-trm } t2) - \{X\}$   
**using** *assms*  
**proof**(*induct t2*)  
**case** (*Susp pi' Y*)  
**hence** *eq: X=Y*  
**unfolding** *occurs.simps(3)* **by** *argo*  
**have** *subst-i: subst [(X, swap pi t1)] (Susp pi' Y) = swap pi'(look-up X [(X, swap pi t1)])*  
**using** *eq subst-susp* **by** *simp*  
**also have**  $\dots = \text{swap } \pi'(\text{swap } \pi t1)$  **by** *simp*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Susp } \pi' Y)) = \text{vars-trm } t1$   
**using** *subst-i swap-append vars-swap* **by** *simp*  
**thus**  
 $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Susp } \pi' Y)) =$   
 $\text{vars-trm } t1 \cup \text{vars-trm } (\text{Susp } \pi' Y) - \{X\}$   
**using** *eq assms(1) not-occurs-subst[of X t1 pi <Susp pi' Y>]* **by** *auto*  
**next**  
**case** (*Paar t21 t22*)  
**hence**  $\text{occurs } X t21 \vee \text{occurs } X t22$   
**unfolding** *occurs.simps(5)* **by** *argo*  
**then show** *?case*  
**proof**  
**assume** *occurs X t21*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] t21) = \text{vars-trm } t1 \cup \text{vars-trm } t21 - \{X\}$   
**using** *assms(1) Paar* **by** *auto*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Paar } t21 t22)) =$   
 $(\text{vars-trm } t1 \cup \text{vars-trm } t21 - \{X\}) \cup (\text{vars-trm } t1 \cup \text{vars-trm } t22 - \{X\})$   
**using** *Paar.hyps(2) assms(1) subst-not-occurs[of X t22 <swap pi t1>]* *not-occurs-trm*  
**by** (*cases occurs X t22*) *auto*  
**thus**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Paar } t21 t22)) =$   
 $\text{vars-trm } t1 \cup \text{vars-trm } (\text{Paar } t21 t22) - \{X\}$  **by** *auto*  
**next**  
**assume** *occurs X t22*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] t22) = \text{vars-trm } t1 \cup \text{vars-trm } t22 - \{X\}$   
**using** *assms(1) Paar* **by** *auto*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Paar } t21 t22)) =$   
 $(\text{vars-trm } t1 \cup \text{vars-trm } t21 - \{X\}) \cup (\text{vars-trm } t1 \cup \text{vars-trm } t22 - \{X\})$   
**using** *Paar.hyps(1) assms(1) subst-not-occurs[of X t21 <swap pi t1>]* *not-occurs-trm*  
**by** (*cases occurs X t21*) *auto*  
**thus**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t1)] (\text{Paar } t21 t22)) =$   
 $\text{vars-trm } t1 \cup \text{vars-trm } (\text{Paar } t21 t22) - \{X\}$  **by** *auto*  
**qed**  
**qed** (*simp-all*)

**lemma** *vars-subseteq:*  
**assumes**  $\neg \text{occurs } X t$

**shows**  $\text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] \text{ } xs) \subseteq (\text{vars-trm } t \cup \text{vars-eprobs } xs)$   
**using** *assms*  
**proof**(*induct xs*)  
**case** *Nil*  
**have**  $\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] [] = []$   
**unfolding** *apply-subst-eprobs-def* **by** *simp*  
**hence**  $\text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] []) = \{\}$   
**by** *simp*  
**thus**  $\text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] []) \subseteq \text{vars-trm } t \cup \text{vars-eprobs } []$   
**by** *simp*  
**next**  
**case** (*Cons a xs*)  
**have**  $\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] (a \# xs) =$   
 $(\text{subst } [(X, \text{swap } \pi t)] (\text{fst } a), \text{subst } [(X, \text{swap } \pi t)] (\text{snd } a)) \# (\text{apply-subst-eprobs}[(X,$   
 $\text{swap } \pi t)] xs)$   
**unfolding** *apply-subst-eprobs-def case-prod-beta* **by** *simp*  
**hence**  $A: \text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] (a \# xs)) =$   
 $(\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)] (\text{snd } a))) \cup (\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)]$   
 $(\text{fst } a)))$   
 $\cup (\text{vars-eprobs } (\text{apply-subst-eprobs}[(X, \text{swap } \pi t)] xs))$   
**by** *auto*  
**then show**  $\text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] (a \# xs))$   
 $\subseteq \text{vars-trm } t \cup \text{vars-eprobs } (a \# xs)$   
**proof**(*cases occurs X (fst a)*)  
**case** *True*  
**have** *X-in-fst:*  
 $\text{occurs } X (\text{fst } a)$  **by** *fact*  
**hence** *vars-fst:*  
 $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)] (\text{fst } a)) = (\text{vars-trm } t \cup \text{vars-trm } (\text{fst } a)) - \{X\}$   
**using** *vars-equ[OF assms X-in-fst]* **by** *simp*  
**then show**  $\text{vars-eprobs } (\text{apply-subst-eprobs } [(X, \text{swap } \pi t)] (a \# xs))$   
 $\subseteq \text{vars-trm } t \cup \text{vars-eprobs } (a \# xs)$   
**proof**(*cases occurs X (snd a)*)  
**case** *True*  
**have** *X-in-snd:*  $\text{occurs } X (\text{snd } a)$  **by** *fact*  
**hence** *vars-snd:*  
 $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)] (\text{snd } a)) = (\text{vars-trm } t \cup \text{vars-trm } (\text{snd } a)) - \{X\}$   
**using** *vars-equ[OF assms X-in-snd]* **by** *simp*  
**with** *vars-fst*  
**have**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)] (\text{snd } a)) \cup \text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)]$   
 $(\text{fst } a)) =$   
 $\text{vars-trm } t \cup \text{vars-trm } (\text{snd } a) \cup \text{vars-trm } (\text{fst } a) - \{X\}$   
**by** *auto*  
**hence**  $\text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)] (\text{snd } a)) \cup \text{vars-trm } (\text{subst } [(X, \text{swap } \pi t)]$   
 $(\text{fst } a)) \subseteq$   
 $\text{vars-trm } t \cup \text{vars-eprobs } (a \# xs)$  **by** *auto*

```

thus ?thesis using A Cons by auto
next
  case False
  hence vars-snd: vars-trm (subst [(X, swap pi t)] (snd a)) = vars-trm (snd a)
    using subst-not-occurs by auto
  with vars-fst
  have vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a)) =
    vars-trm t  $\cup$  vars-trm (snd a)  $\cup$  vars-trm (fst a) - {X}
    using False not-occurs-trm by auto
  hence vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a))  $\subseteq$ 
    vars-trm t  $\cup$  vars-eprobs (a # xs) by auto
  thus ?thesis using A Cons by auto
qed
next
  case False
  hence vars-fst:
    vars-trm (subst [(X, swap pi t)] (fst a)) = vars-trm (fst a)
    using subst-not-occurs by simp
  then show vars-eprobs (apply-subst-eprobs [(X, swap pi t)] (a # xs))
 $\subseteq$  vars-trm t  $\cup$  vars-eprobs (a # xs)
  proof(cases occurs X (snd a))
    case True
    have X-in-snd: occurs X (snd a) by fact
    hence vars-snd:
      vars-trm (subst [(X, swap pi t)] (snd a)) = (vars-trm t  $\cup$  vars-trm (snd
a)) - {X}
      using vars-equ[OF assms X-in-snd] by simp
    with vars-fst
    have vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a)) =
      vars-trm t  $\cup$  vars-trm (snd a)  $\cup$  vars-trm (fst a) - {X}
      using False not-occurs-trm by auto
    hence vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a))  $\subseteq$ 
      vars-trm t  $\cup$  vars-eprobs (a # xs) by auto
    thus ?thesis using A Cons by auto
  next
  case False
  hence vars-snd: vars-trm (subst [(X, swap pi t)] (snd a)) = vars-trm (snd a)
    using subst-not-occurs by auto
  with vars-fst
  have vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a)) =
    vars-trm (snd a)  $\cup$  vars-trm (fst a)
    by simp
  hence vars-trm (subst [(X, swap pi t)] (snd a))  $\cup$  vars-trm (subst [(X, swap
pi t)] (fst a))  $\subseteq$ 

```

```

      vars-trm t ∪ vars-eprobs (a # xs) by auto
    thus ?thesis using A Cons by auto
  qed
qed
qed

lemma vars-decrease:
  assumes ¬occurs X t
  shows card (vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs))
    < card (insert X (vars-trm t ∪ vars-eprobs xs))
  using assms
proof(cases X ∈ (vars-trm t ∪ vars-eprobs xs))
  case True
  hence insert X (vars-trm t ∪ vars-eprobs xs) = vars-trm t ∪ vars-eprobs xs
    using insert-absorb by auto
  moreover have subset:
    vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs) ⊆ vars-trm t ∪ vars-eprobs
  xs
  using assms vars-subseteq by simp
  hence
    vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs) ⊆ vars-trm t ∪ vars-eprobs
  xs - {X}
  using not-occurs-list assms subset-Diff-insert by auto
  ultimately have
    card (vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs))
      ≤ card (vars-trm t ∪ vars-eprobs xs - {X})
    using Finite-Set.card-mono
  by (metis finite-Diff finite-Un vars-eprobs-finite vars-term-finite)
  thus card (vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs))
    < card (insert X (vars-trm t ∪ vars-eprobs xs))
    by (simp add: card.insert-remove)
next
  case False
  hence card (vars-trm t ∪ vars-eprobs xs) < card (insert X (vars-trm t ∪ vars-eprobs
  xs))
    by auto
  moreover have subset:
    vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs) ⊆ vars-trm t ∪ vars-eprobs
  xs
  using assms vars-subseteq by simp
  hence card (vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs))
    ≤ card (vars-trm t ∪ vars-eprobs xs)
    using Finite-Set.card-mono
  by (metis finite-Un vars-eprobs-finite vars-term-finite)
  ultimately show
    card (vars-eprobs (apply-subst-eprobs [(X, swap pi t)] xs))
      < card (insert X (vars-trm t ∪ vars-eprobs xs))
    by simp

```

qed

**lemma** *rank-r-cred*:

**assumes**  $P1 \vdash (nabla) \rightarrow P2$

**shows**  $(P2, P1) \in \text{rank-r}$

**unfolding** *rank-r-def* **by** (*cases rule: c-red.cases*[*OF*  $\langle P1 \vdash nabla \rightarrow P2 \rangle$ ], *simp-all*)

**lemma** *rank-r-sred*:

**assumes**  $P1 \vdash s \rightsquigarrow P2$

**shows**  $(P2, P1) \in \text{rank-r}$

**proof**(*cases rule: s-red.cases*[*OF*  $\langle P1 \vdash s \rightsquigarrow P2 \rangle$ ])

**case** ( $\ell$   $t1$   $t2$   $s1$   $s2$   $xs$   $ys$ )

**case** ( $\ell$   $t1$   $t2$   $s1$   $s2$   $xs$   $ys$ )

**let**  $?union = \text{vars-trm } s1 \cup \text{vars-trm } s2 \cup \text{vars-trm } t1 \cup \text{vars-trm } t2 \cup \text{vars-eprobs } xs$

**and**  $?size = \text{size-trm } t1 + \text{size-trm } t2 + \text{size-trm } s1 + \text{size-trm } s2 + \text{size-eprobs } xs$

**from**  $\ell$  **have**  $\text{vars-eprobs } ((\text{Paar } t1 \ t2, \text{Paar } s1 \ s2) \# xs) = ?union$

**unfolding** *vars-eprobs.simps vars-trm.simps* **by** *auto*

**moreover from**  $\ell$  **have**

$\text{size-eprobs } ((\text{Paar } t1 \ t2, \text{Paar } s1 \ s2) \# xs) = \ell + ?size$

**unfolding** *size-eprobs.simps* **using** *size-trm.simps(6)* **by** *auto*

**from**  $\ell$  **have**

$\text{vars-eprobs } ((t1, s1) \# (t2, s2) \# xs) = ?union$

**unfolding** *vars-eprobs.simps* **by** *auto*

**moreover from**  $\ell$  **have**

$\text{size-eprobs } ((t1, s1) \# (t2, s2) \# xs) = ?size$

**unfolding** *size-eprobs.simps* **by** *simp*

**ultimately have**

$\text{size-eprobs } ((t1, s1) \# (t2, s2) \# xs) < \text{size-eprobs } ((\text{Paar } t1 \ t2, \text{Paar } s1 \ s2) \# xs)$

$\text{card } (\text{vars-eprobs } ((t1, s1) \# (t2, s2) \# xs)) = \text{card } (\text{vars-eprobs } ((\text{Paar } t1 \ t2, \text{Paar } s1 \ s2) \# xs))$

**by** *simp+*

**with**  $\ell$  **show**  $(P2, P1) \in \text{rank-r}$

**unfolding** *rank-r-def* **by** *simp*

**next**

**case** ( $\gamma$   $pi1$   $X$   $pi2$   $xs$   $ys$ )

**let**  $?mapds = \text{map } (\lambda a. (a, \text{Susp } [] \ X)) (\text{ds-list } pi1 \ pi2) @ ys$

**let**  $?union = \{X\} \cup \text{vars-eprobs } xs$

**from**  $\gamma$  **have**

$\text{vars: vars-eprobs } ((\text{Susp } pi1 \ X, \text{Susp } pi2 \ X) \# xs) = ?union$  **and**

$\text{size: size-eprobs } ((\text{Susp } pi1 \ X, \text{Susp } pi2 \ X) \# xs) = \ell + \text{size-eprobs } xs$

**unfolding** *vars-eprobs.simps size-eprobs.simps* **by** *simp+*

**then show**  $(P2, P1) \in \text{rank-r}$

**proof**(*cases*  $X \in \text{vars-eprobs } xs$ )

**case** *True*

```

hence  $\text{card } ?\text{union} = \text{card } (\text{vars-eprobs } xs)$ 
  by (simp add: insert-absorb)
with 7 show  $(P2, P1) \in \text{rank-}r$ 
  unfolding rank-r-def by simp
next
  case False
  hence  $\text{card } ?\text{union} = 1 + \text{card } (\text{vars-eprobs } xs)$ 
  by auto
  with 7 show  $(P2, P1) \in \text{rank-}r$ 
  unfolding rank-r-def by simp
qed
next
case ( $8 X t pi xs ys$ )
  let  $?\text{union} = \text{insert } X (\text{vars-trm } t \cup \text{vars-eprobs } xs)$ 
  and  $?size = \text{size-trm } t + \text{size-eprobs } xs$ 
  from 8 have
     $\text{vars: vars-eprobs } ((\text{Susp } pi X, t) \# xs) = ?\text{union}$  and
     $\text{size: size-eprobs } ((\text{Susp } pi X, t) \# xs) = 1 + ?size$ 
  unfolding vars-eprobs.simps size-eprobs.simps by simp+
  moreover from 8 have
     $P2 = (\text{apply-subst-eprobs } [(X, \text{swap } (\text{rev } pi) t)] xs,$ 
     $\text{apply-subst-fprobs } [(X, \text{swap } (\text{rev } pi) t)] ys)$ 
    using apply-subst-equivalence by auto
  ultimately show  $?thesis$ 
  using 8 vars-decrease[OF 8(4)] unfolding rank-r-def by simp
next
case ( $9 X t pi xs ys$ )
  let  $?\text{union} = \text{insert } X (\text{vars-trm } t \cup \text{vars-eprobs } xs)$ 
  and  $?size = \text{size-trm } t + \text{size-eprobs } xs$ 
  from 9 have
     $\text{vars: vars-eprobs } ((t, \text{Susp } pi X) \# xs) = ?\text{union}$  and
     $\text{size: size-eprobs } ((t, \text{Susp } pi X) \# xs) = 1 + ?size$ 
  unfolding vars-eprobs.simps size-eprobs.simps by simp+
  moreover from 9 have
     $P2 = (\text{apply-subst-eprobs } [(X, \text{swap } (\text{rev } pi) t)] xs,$ 
     $\text{apply-subst-fprobs } [(X, \text{swap } (\text{rev } pi) t)] ys)$ 
    using apply-subst-equivalence by auto
  ultimately show  $?thesis$ 
  using 9 vars-decrease[OF 9(4)] unfolding rank-r-def by simp
qed (unfold rank-r-def, auto simp add: vars-swap)

lemma rank-r-trans:  $[(P1, P2) \in \text{rank-}r; (P2, P3) \in \text{rank-}r] \implies (P1, P3) \in \text{rank-}r$ 
  unfolding rank-r-def by auto

lemma rank-r-red-plus:
  assumes  $P1 \models (s, nabla) \Rightarrow P2$ 
  shows  $(P2, P1) \in \text{rank-}r$ 
  using assms
by (induct rule: red-plus.induct) (auto dest: rank-r-sred rank-r-cred rank-r-trans)

```

**lemma** *wf-rank-r*:  
**shows** *wf (rank-r)*  
**unfolding** *rank-r-def* **by** *simp*

## 12 Unification

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

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

**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!*]: *fail* ( $\llbracket \text{[]}, a \# ? \text{Atom } a \# \text{ys} \rrbracket$ )  
*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!*]: *fail* ( $\text{Abst } a \approx ? \text{Unit} \# \text{xs}, \text{ys}$ )  
*fail-abst-atom* [*intro!*]: *fail* ( $\text{Abst } a \approx ? \text{Atom } b \# \text{xs}, \text{ys}$ )  
*fail-abst-paar* [*intro!*]: *fail* ( $\text{Abst } a \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}$ )  
*fail-func-abst* [*intro!*]: *fail* ( $\text{Func } F \ t1 \approx ? \text{Abst } a \ t \# \text{xs}, \text{ys}$ )  
*fail-atom-unit* [*intro!*]: *fail* ( $\text{Atom } b \approx ? \text{Unit} \# \text{xs}, \text{ys}$ )  
*fail-paar-unit* [*intro!*]: *fail* ( $\text{Paar } t1 \ t2 \approx ? \text{Unit} \# \text{xs}, \text{ys}$ )  
*fail-func-unit* [*intro!*]: *fail* ( $\text{Func } F \ t1 \approx ? \text{Unit} \# \text{xs}, \text{ys}$ )  
*fail-atom-paar* [*intro!*]: *fail* ( $\text{Atom } a \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}$ )  
*fail-func-atom* [*intro!*]: *fail* ( $\text{Func } F \ t1 \approx ? \text{Atom } a \# \text{xs}, \text{ys}$ )  
*fail-func-paar* [*intro!*]: *fail* ( $\text{Func } F \ t \approx ? \text{Paar } t1 \ t2 \# \text{xs}, \text{ys}$ )  
*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!*]: *fail* ( $s \approx ? t \# \text{xs}, \text{ys}$ )  $\Longrightarrow$  *fail* ( $t \approx ? s \# \text{xs}, \text{ys}$ )

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

**lemma** *U-equ-symm*:  
**shows**  $U (s \approx ? t \# \text{xs}, \text{ys}) = U (t \approx ? s \# \text{xs}, \text{ys})$   
**by** (*auto simp add: all-solutions-def equ-symm*)

```

lemma fail-then-empty:
  assumes fail P1
  shows  $U P1 = \{\}$ 
  using assms
proof(induct rule: fail.induct)
  case (fail-occur-abst X t pi a xs ys)
  let ?P = (Susp pi X  $\approx?$  Abst a t # xs, ys)
  { assume  $U ((Susp pi X, Abst a t) \# xs, ys) \neq \{\}$ 
    then obtain s nabra where eq1:  $nabra \vdash subst s (Susp pi X) \approx Abst a (subst s t)$ 
      by (auto simp add: all-solutions-def)
    moreover
    have occurs X t by fact
    then obtain t' pi' where
      eq2:  $nabra \vdash subst s (Susp pi X) \approx swap pi' t' t' \in sub-trms (subst s t)$ 
      using occurs-sub-trm-equ by blast
    moreover
    have eq3:  $\neg nabra \vdash (Abst a (subst s t)) \approx swap pi' t'$ 
      using eq2 psub-trm-not-equ by auto
    then have False using eq1 eq2 eq3
      by (metis equ-symm equ-trans)
  }
  then show  $U ?P = \{\}$  by auto
next
  case (fail-occur-func X t pi F xs ys)
  let ?P = (Susp pi X  $\approx?$  Func F t # xs, ys)
  { assume  $U ((Susp pi X, Func F t) \# xs, ys) \neq \{\}$ 
    then obtain s nabra where eq1:  $nabra \vdash subst s (Susp pi X) \approx Func F (subst s t)$ 
      by (auto simp add: all-solutions-def)
    moreover
    have occurs X t by fact
    then obtain t' pi' where
      eq2:  $nabra \vdash subst s (Susp pi X) \approx swap pi' t' t' \in sub-trms (subst s t)$ 
      using occurs-sub-trm-equ by blast
    moreover
    have eq3:  $\neg nabra \vdash (Func F (subst s t)) \approx swap pi' t'$ 
      using eq2 psub-trm-not-equ by auto
    then have False using eq1 eq2 eq3
      by (metis equ-symm equ-trans)
  }
  then show  $U ?P = \{\}$  by auto
next
  case (fail-occur-paar X t1 t2 pi xs ys)
  let ?P = (Susp pi X  $\approx?$  Paar t1 t2 # xs, ys)
  have occurs X t1  $\vee$  occurs X t2 by fact

```

```

then show  $U \text{ ?}P = \{\}$ 
proof
  {assume occurs  $X \ t1$ 
   {assume  $U ((Susp \ pi \ X, \ Paar \ t1 \ t2) \ \# \ xs, \ ys) \neq \{\}$ 
   then obtain  $s \ nabla$  where  $eq1: \ nabla \vdash \ subst \ s \ (Susp \ pi \ X) \approx \ Paar \ (subst \ s \ t1) \ (subst \ s \ t2)$ 
   by (auto simp add: all-solutions-def)
  moreover
  have occurs  $X \ t1$  by fact
  then obtain  $t' \ pi'$  where
     $eq2: \ nabla \vdash \ subst \ s \ (Susp \ pi \ X) \approx \ swap \ pi' \ t' \ t' \in \ sub\text{-}trms \ (subst \ s \ t1)$ 
    using occurs-sub-trm-equ by blast
  moreover
  have  $eq3: \neg \ nabla \vdash \ (Paar \ (subst \ s \ t1) \ (subst \ s \ t2)) \approx \ swap \ pi' \ t'$ 
    using  $eq2$  psub-trm-not-equ by auto
  then have False using  $eq1 \ eq2 \ eq3$ 
    by (metis equ-symm equ-trans)}
then show  $U \text{ ?}P = \{\}$  by auto}

  {assume occurs  $X \ t2$ 
   {assume  $U ((Susp \ pi \ X, \ Paar \ t1 \ t2) \ \# \ xs, \ ys) \neq \{\}$ 
   then obtain  $s \ nabla$  where  $eq1: \ nabla \vdash \ subst \ s \ (Susp \ pi \ X) \approx \ Paar \ (subst \ s \ t1) \ (subst \ s \ t2)$ 
   by (auto simp add: all-solutions-def)
  moreover
  have occurs  $X \ t2$  by fact
  then obtain  $t' \ pi'$  where
     $eq2: \ nabla \vdash \ subst \ s \ (Susp \ pi \ X) \approx \ swap \ pi' \ t' \ t' \in \ sub\text{-}trms \ (subst \ s \ t2)$ 
    using occurs-sub-trm-equ by blast
  moreover
  have  $eq3: \neg \ nabla \vdash \ (Paar \ (subst \ s \ t1) \ (subst \ s \ t2)) \approx \ swap \ pi' \ t'$ 
    using  $eq2$  psub-trm-not-equ by auto
  then have False using  $eq1 \ eq2 \ eq3$ 
    by (metis equ-symm equ-trans)
  }
  then show  $U \text{ ?}P = \{\}$  by auto}
qed
next
case (fail-fresh-atom  $a \ ys$ )
let  $\text{?}P = (\ [], \ a \ \# \ ? \ Atom \ a \ \# \ ys)$ 
have  $\nabla \ nabla \ s. \ nabla \vdash \ a \ \# \ subst \ s \ (Atom \ a)$ 
  using subst-atom Fresh-elim3 by auto
thus  $U \text{ ?}P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-diff-atoms  $a \ b \ xs \ ys$ )
let  $\text{?}P = (Atom \ a \ \approx \ ? \ Atom \ b \ \# \ xs, \ ys)$ 
from  $\langle a \neq b \rangle$  have  $\nabla \ nabla \ s. \ nabla \vdash \ subst \ s \ (Atom \ a) \approx \ subst \ s \ (Atom \ b)$ 
  using Equ-elim1 by auto

```

```

thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
  case (fail-abst-unit  $a \ t \ xs \ ys$ )
  let  $?P = (Abst \ a \ t \ \approx? \ Unit \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Abst \ a \ t) \ \approx \ subst \ s \ Unit$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-abst-atom  $a \ t \ b \ xs \ ys$ )
  let  $?P = (Abst \ a \ t \ \approx? \ Atom \ b \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Abst \ a \ t) \ \approx \ subst \ s \ (Atom \ b)$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-abst-paar  $a \ t \ t1 \ t2 \ xs \ ys$ )
  let  $?P = (Abst \ a \ t \ \approx? \ Paar \ t1 \ t2 \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Abst \ a \ t) \ \approx \ subst \ s \ (Paar \ t1 \ t2)$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-func-abst  $F \ t1 \ a \ t \ xs \ ys$ )
  let  $?P = (Func \ F \ t1 \ \approx? \ Abst \ a \ t \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Func \ F \ t1) \ \approx \ subst \ s \ (Abst \ a \ t)$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-atom-unit  $b \ xs \ ys$ )
  let  $?P = (Atom \ b \ \approx? \ Unit \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Atom \ b) \ \approx \ subst \ s \ (Unit)$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-paar-unit  $t1 \ t2 \ xs \ ys$ )
  let  $?P = (Paar \ t1 \ t2 \ \approx? \ Unit \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Paar \ t1 \ t2) \ \approx \ subst \ s \ (Unit)$ 
    by (auto elim: equ.cases)
  thus  $U \ ?P = \{\}$ 
    using all-solutions-def by simp
next
  case (fail-func-unit  $F \ t1 \ xs \ ys$ )
  let  $?P = (Func \ F \ t1 \ \approx? \ Unit \ \# \ xs, \ ys)$ 
  have  $\# \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Func \ F \ t1) \ \approx \ subst \ s \ (Unit)$ 
    by (auto elim: equ.cases)

```

```

thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-atom-paar  $b \ t1 \ t2 \ xs \ ys$ )
let  $?P = (Atom \ b \ \approx? \ Paar \ t1 \ t2 \ \# \ xs, \ ys)$ 
have  $\nabla \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Atom \ b) \ \approx \ subst \ s \ (Paar \ t1 \ t2)$ 
  by (auto elim: equ.cases)
thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-func-atom  $F \ t1 \ b \ xs \ ys$ )
let  $?P = (Func \ F \ t1 \ \approx? \ Atom \ b \ \# \ xs, \ ys)$ 
have  $\nabla \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Func \ F \ t1) \ \approx \ subst \ s \ (Atom \ b)$ 
  by (auto elim: equ.cases)
thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-func-paar  $F \ t \ t1 \ t2 \ xs \ ys$ )
let  $?P = (Func \ F \ t \ \approx? \ Paar \ t1 \ t2 \ \# \ xs, \ ys)$ 
have  $\nabla \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Func \ F \ t) \ \approx \ subst \ s \ (Paar \ t1 \ t2)$ 
  by (auto elim: equ.cases)
thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-diff-func  $F1 \ F2 \ t1 \ t2 \ xs \ ys$ )
let  $?P = (Func \ F1 \ t1 \ \approx? \ Func \ F2 \ t2 \ \# \ xs, \ ys)$ 
from  $\langle F1 \neq F2 \rangle$  have  $\nabla \ nabla \ s. \ nabla \ \vdash \ subst \ s \ (Func \ F1 \ t1) \ \approx \ subst \ s \ (Func \ F2 \ t2)$ 
  by (auto elim: equ.cases)
thus  $U \ ?P = \{\}$ 
  using all-solutions-def by simp
next
case (fail-sym  $s \ t \ xs \ ys$ )
let  $?P = (t \ \approx? \ s \ \# \ xs, \ ys)$ 
have fail  $((s, t) \ \# \ xs, \ ys)$ 
   $U \ ((s, t) \ \# \ xs, \ ys) = \{\}$  by fact+
thus  $U \ ?P = \{\}$ 
  using all-solutions-def U-equ-symm by simp
qed

```

**lemma** *not-reduce-then-fail*:

```

assumes  $\neg (\exists \ nabla \ s \ P'. ((t1 \ \approx? \ t2) \ \# \ xs, \ ys) \models (nabla, s) \Rightarrow P')$ 
shows fail  $((t1 \ \approx? \ t2) \ \# \ xs, \ ys)$ 
using assms
proof(cases t1)
  case (Abst a t1')

```

```

have t1-def: t1 = Abst a t1' by fact
then show fail ((t1, t2) # xs, ys)
proof(cases t2)
  case (Abst b t2')
  with t1-def
  have ((t1 ≈? t2)#xs,ys) ⊢[] ~> ((t1' ≈?t2')#xs,ys) ∨
  ((t1 ≈? t2)#xs,ys) ⊢[] ~> ((t1' ≈?swap [(a,b)] t2')#xs,(a#?t2')#ys)
  using abst-aa-sred abst-ab-sred by (cases a=b) auto
  hence ∃ P2. ((t1 ≈? t2)#xs,ys) ⊢({},[]) ⇒P2
  using sred-single by blast
  thus fail ((t1, t2) # xs, ys)
  using assms by simp
next
case (Susp pi X)
have t2-def: t2 = Susp pi X by fact
with t1-def
show fail ((t1, t2) # xs, ys)
proof(cases occurs X t1')
  case True
  with t1-def t2-def
  show fail ((t1, t2) # xs, ys)
  using fail-sym[OF fail-occur-abst[OF True]] by simp
next
case False
with t1-def
have not-occurs: ¬ occurs X t1 by simp
hence ((t1 ≈? Susp pi X)#xs,ys)
  ⊢[(X,swap (rev pi) t1)] ~> apply-subst [(X,swap (rev pi) t1)] (xs,ys)
  using t1-def var-2-sred[OF not-occurs] by simp
  hence ∃ P2 s. ((t1 ≈? t2)#xs,ys) ⊢({},s) ⇒P2
  using t1-def t2-def sred-single by blast
  thus fail ((t1, t2) # xs, ys)
  using assms by simp
qed
qed (auto)
next
case (Susp pi X)
have t1-def: t1 = Susp pi X by fact
then show fail ((t1, t2) # xs, ys)
proof(cases occurs X t2)
  case True
  then show fail ((t1, t2) # xs, ys)
  proof(cases t2)
    case (Abst a t2')
    have t2-def: t2 = Abst a t2' by fact
    with True
    have occurs X t2' unfolding occurs.simps(4) t2-def by argo
    thus fail ((t1, t2) # xs, ys)
    using t1-def t2-def fail-occur-abst by simp

```

```

next
  case (Susp pi' Y)
  have t2-def: t2 = Susp pi' Y by fact
  have X = Y
    using True unfolding t2-def occurs.simps(3)
    by argo
  hence ((Susp pi X ≈? Susp pi' Y) # xs, ys)
    ⊢ [] ~> (xs, (map (λa. a#? Susp [] X) (ds-list pi pi')) @ ys)
    using susp-sred by simp
  hence ∃ P2. ((t1 ≈? t2) # xs, ys) ⊨ ({}, []) ⇒ P2
    using sred-single t1-def t2-def by blast
  thus fail ((t1, t2) # xs, ys)
    using assms by simp
next
  case (Paar t21 t22)
  have t2-def: t2 = Paar t21 t22 by fact
  with True
  have occurs X t21 ∨ occurs X t22 unfolding occurs.simps(5) t2-def by argo
  thus fail ((t1, t2) # xs, ys)
    using t1-def t2-def fail-occur-paar by simp
next
  case (Func f t2')
  have t2-def: t2 = Func f t2' by fact
  with True
  have occurs X t2' unfolding occurs.simps(6) t2-def by argo
  thus fail ((t1, t2) # xs, ys)
    using t1-def t2-def fail-occur-func by simp
qed (auto simp add: True)
next
  case False
  hence ((Susp pi X, t2) # xs, ys) ⊢
    [(X, swap (rev pi) t2)] ~> apply-subst [(X, swap (rev pi) t2)] (xs, ys)
    using var-1-sred[OF False] by simp
  hence ∃ P2 s. ((t1 ≈? t2) # xs, ys) ⊨ ({}, s) ⇒ P2
    using t1-def sred-single by blast
  thus fail ((t1, t2) # xs, ys)
    using assms by simp
qed
next
  case Unit
  have t1-def: t1 = Unit by fact
  then show fail ((t1, t2) # xs, ys)
  proof (cases t2)
    case (Susp pi X)
    have t2-def: t2 = Susp pi X by fact
    with t1-def have not-occurs: ¬ occurs X t1 by simp
    hence ((t1 ≈? Susp pi X) # xs, ys)
      ⊢ [(X, swap (rev pi) t1)] ~> apply-subst [(X, swap (rev pi) t1)] (xs, ys)
      using t1-def var-2-sred[OF not-occurs] by simp
  
```

```

hence  $\exists P2 s. ((t1 \approx? t2) \# xs, ys) \models (\{ \}, s) \Rightarrow P2$ 
  using t1-def t2-def sred-single by blast
thus fail  $((t1, t2) \# xs, ys)$ 
  using assms by simp
next
case Unit
with t1-def
have  $((t1 \approx? t2) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$ 
  using unit-sred by auto
hence  $\exists P2. ((t1 \approx? t2) \# xs, ys) \models (\{ \}, []) \Rightarrow P2$ 
  using sred-single by blast
thus fail  $((t1, t2) \# xs, ys)$ 
  using assms by simp
qed (auto)
next
case (Atom a)
have t1-def:  $t1 = Atom a$  by fact
then show fail  $((t1, t2) \# xs, ys)$ 
proof(cases t2)
  case (Susp pi X)
  have t2-def:  $t2 = Susp pi X$  by fact
  with t1-def have not-occurs:  $\neg occurs X t1$  by simp
  hence  $((t1 \approx? Susp pi X) \# xs, ys)$ 
     $\vdash [(X, swap (rev pi) t1)] \rightsquigarrow apply-subst [(X, swap (rev pi) t1)] (xs, ys)$ 
    using t1-def var-2-sred[OF not-occurs] by simp
  hence  $\exists P2 s. ((t1 \approx? t2) \# xs, ys) \models (\{ \}, s) \Rightarrow P2$ 
    using t1-def t2-def sred-single by blast
  thus fail  $((t1, t2) \# xs, ys)$ 
    using assms by simp
next
case (Atom b)
have t2-def:  $t2 = Atom b$  by fact
then show fail  $((t1, t2) \# xs, ys)$ 
proof(cases  $a=b$ )
  case True
  hence  $((t1 \approx? t2) \# xs, ys) \vdash [] \rightsquigarrow (xs, ys)$ 
    using t2-def t1-def atom-sred by simp
  hence  $\exists P2. ((t1 \approx? t2) \# xs, ys) \models (\{ \}, []) \Rightarrow P2$ 
    using sred-single by blast
  thus fail  $((t1, t2) \# xs, ys)$ 
    using assms by simp
  qed (simp add: t1-def t2-def fail-diff-atoms)
qed(auto)
next
case (Paar t11 t12)
have t1-def:  $t1 = Paar t11 t12$  by fact
then show fail  $((t1, t2) \# xs, ys)$ 
proof(cases t2)
next

```



```

hence ((t1 ≈? Susp pi X) # xs, ys)
  ⊢ [(X, swap (rev pi) t1)] ~> apply-subst [(X, swap (rev pi) t1)] (xs, ys)
using t1-def var-2-sred[OF not-occurs] by simp
hence ∃ P2 s. ((t1 ≈? t2) # xs, ys) ⊢ ({}, s) ⇒ P2
using t1-def t2-def sred-single by blast
thus fail ((t1, t2) # xs, ys)
using assms by simp
qed
next
case (Func g t2')
have t2-def: t2 = Func g t2' by fact
then show fail ((t1, t2) # xs, ys)
proof(cases f=g)
  case True
hence ((t1 ≈? t2) # xs, ys) ⊢ [] ~> ((t1' ≈? t2') # xs, ys)
using t1-def t2-def func-sred by simp
hence ∃ P2. ((t1 ≈? t2) # xs, ys) ⊢ ({}, []) ⇒ P2
using sred-single by blast
thus fail ((t1, t2) # xs, ys)
using assms by simp
next
case False
then show fail ((t1, t2) # xs, ys)
using t1-def t2-def fail-diff-func[OF False] by simp
qed
qed(auto)
qed

```

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

```

assumes t ≠ Atom a
shows ∃ P2 nabra s. ([], (a #? t) # xs) ⊢ (nabra, s) ⇒ P2
using assms cred-single
proof(cases t)
  case (Abst b t')
then show ∃ P2 nabra s. ([], (a #? t) # xs) ⊢ (nabra, s) ⇒ P2
proof(cases a=b)
  case True
hence ([], (a #? t) # xs) ⊢ {} → ([], xs)
unfolding Abst using abst-aa-cred by simp
then show ∃ P2 nabra s. ([], (a #? t) # xs) ⊢ (nabra, s) ⇒ P2
using cred-single by blast
next
case False
hence ([], (a #? t) # xs) ⊢ {} → ([], (a #? t') # xs)
unfolding Abst using abst-ab-cred by simp
then show ∃ P2 nabra s. ([], (a #? t) # xs) ⊢ (nabra, s) ⇒ P2
using cred-single by blast
qed

```

**next**  
**case** (*Atom b*)  
**with** *assms*  
**have**  $a \neq b$  **by** *simp*  
**hence**  $([], (a \#? t) \# xs) \vdash \{\} \rightarrow ([], xs)$   
**unfolding** *Atom* **using** *atom-cred* **by** *simp*  
**then show**  $\exists P2 \text{ nabl} a s. ( [], (a \#? t) \# xs) \models (\text{nabl} a, s) \Rightarrow P2$   
**using** *cred-single* **by** *blast*  
**qed** (*simp add: cred-single, blast+*)

**lemma** *empty-stuck*:  
**shows**  $([], []) \in \text{stuck}$   
**proof** –  
**have**  $\neg (\exists P2 \text{ nabl} a s. ( [], []) \models (\text{nabl} a, s) \Rightarrow P2)$   
**by** (*auto elim: red-plus.cases s-red.cases c-red.cases*)  
**thus**  $([], []) \in \text{stuck}$   
**unfolding** *stuck-def* **by** *auto*  
**qed**

**lemma** *fail-is-stuck*:  
**assumes** *fail P*  
**shows**  $P \in \text{stuck}$   
**using** *assms*  
**proof** (*induct rule: fail.induct*)  
**case** (*fail-occur-abst X t pi a xs ys*)  
**have** *t-occurs: occurs X t* **by** *fact*  
**moreover have**  $\neg (\exists P2 \text{ nabl} a s. ((\text{Susp } \text{pi } X \approx? \text{Abst } a t) \# xs, ys) \models (\text{nabl} a, s) \Rightarrow P2)$   
**proof**  
**assume**  $\exists P2 \text{ nabl} a s. ((\text{Susp } \text{pi } X \approx? \text{Abst } a t) \# xs, ys) \models (\text{nabl} a, s) \Rightarrow P2$   
**then obtain**  $P2 \text{ nabl} a s$  **where**  
 $\text{red}: ((\text{Susp } \text{pi } X \approx? \text{Abst } a t) \# xs, ys) \models (\text{nabl} a, s) \Rightarrow P2$   
**by** *auto*  
**thus** *False*  
**proof** (*cases rule: red-plus.cases*)  
**case** *sred-single*  
**have**  $((\text{Susp } \text{pi } X, \text{Abst } a t) \# xs, ys) \vdash s \rightsquigarrow P2$  **by** *fact*  
**hence**  $\neg \text{occurs } X t$   
**by** (*auto elim: s-red.cases*)  
**with** *t-occurs* **show** *False* **by** *simp*  
**next**  
**case** *cred-single*  
**have**  $((\text{Susp } \text{pi } X, \text{Abst } a t) \# xs, ys) \vdash \text{nabl} a \rightarrow P2$  **by** *fact*  
**moreover have**  $\text{fst } ((\text{Susp } \text{pi } X, \text{Abst } a t) \# xs, ys) \neq []$   
**by** *simp*  
**ultimately show** *False*  
**using** *c-red-eqs-empty* **by** *blast*  
**next**

```

    case (sred-step s1 P3 s2)
    have ((Susp pi X, Abst a t) # xs, ys) ⊢ s1 ~> P3 by fact
    hence ¬ occurs X t
      by (auto elim: s-red.cases)
    with t-occurs show False by simp
  next
    case (cred-step nabla1 P3 nabla2)
    have ((Susp pi X, Abst a t) # xs, ys) ⊢ nabla1 → P3 by fact
    moreover have fst ((Susp pi X, Abst a t) # xs, ys) ≠ []
      by simp
    ultimately show False
      using c-red-eqs-empty by blast
  qed
  qed
  then show ((Susp pi X, Abst a t) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
  next
    case (fail-occur-func X t pi F xs ys)
    have t-occurs: occurs X t by fact
    moreover have ¬ (∃ P2 nabla s. ((Susp pi X ≈? Func F t) # xs, ys) ⊨ (nabla,s)
    ⇒ P2)
  proof
    assume ∃ P2 nabla s. ((Susp pi X ≈? Func F t) # xs, ys) ⊨ (nabla,s) ⇒ P2
    then obtain P2 nabla s where
      red: ((Susp pi X ≈? Func F t) # xs, ys) ⊨ (nabla,s) ⇒ P2
      by auto
    thus False
  proof (cases rule: red-plus.cases)
    case sred-single
    have ((Susp pi X, Func F t) # xs, ys) ⊢ s ~> P2 by fact
    hence ¬ occurs X t
      by (auto elim: s-red.cases)
    with t-occurs show False by simp
  next
    case cred-single
    have ((Susp pi X, Func F t) # xs, ys) ⊢ nabla → P2 by fact
    moreover have fst ((Susp pi X, Func F t) # xs, ys) ≠ []
      by simp
    ultimately show False
      using c-red-eqs-empty by blast
  next
    case (sred-step s1 P3 s2)
    have ((Susp pi X, Func F t) # xs, ys) ⊢ s1 ~> P3 by fact
    hence ¬ occurs X t
      by (auto elim: s-red.cases)
    with t-occurs show False by simp
  next
    case (cred-step nabla1 P3 nabla2)
    have ((Susp pi X, Func F t) # xs, ys) ⊢ nabla1 → P3 by fact

```

```

    moreover have fst ((Susp pi X, Func F t) # xs, ys) ≠ []
      by simp
    ultimately show False
      using c-red-eqs-empty by blast
  qed
qed
then show ((Susp pi X, Func F t) # xs, ys) ∈ stuck
  unfolding stuck-def by simp
next
case (fail-occur-paar X t1 t2 pi xs ys)
have occurs X t1 ∨ occurs X t2 by fact
hence t-occurs: occurs X (Paar t1 t2)
  using occurs.simps(5) by auto
moreover have ¬ (∃ P2 nabra s. ((Susp pi X ≈? Paar t1 t2) # xs, ys) ⊨
(nabra,s) ⇒ P2)
proof
  assume ∃ P2 nabra s. ((Susp pi X ≈? Paar t1 t2) # xs, ys) ⊨ (nabra,s) ⇒ P2
  then obtain P2 nabra s where
  red: ((Susp pi X ≈? Paar t1 t2) # xs, ys) ⊨ (nabra,s) ⇒ P2
    by auto
  thus False
proof (cases rule: red-plus.cases)
  case sred-single
  have ((Susp pi X, Paar t1 t2) # xs, ys) ⊢ s ∼ P2 by fact
  hence ¬ occurs X (Paar t1 t2)
    by (auto elim: s-red.cases)
  with t-occurs show False by simp
next
case cred-single
  have ((Susp pi X, Paar t1 t2) # xs, ys) ⊢ nabra → P2 by fact
  moreover have fst ((Susp pi X, Paar t1 t2) # xs, ys) ≠ []
    by simp
  ultimately show False
    using c-red-eqs-empty by blast
next
case (sred-step s1 P3 s2)
  have ((Susp pi X, Paar t1 t2) # xs, ys) ⊢ s1 ∼ P3 by fact
  hence ¬ occurs X (Paar t1 t2)
    by (auto elim: s-red.cases)
  with t-occurs show False by simp
next
case (cred-step nabra1 P3 nabra2)
  have ((Susp pi X, Paar t1 t2) # xs, ys) ⊢ nabra1 → P3 by fact
  moreover have fst ((Susp pi X, Paar t1 t2) # xs, ys) ≠ []
    by simp
  ultimately show False
    using c-red-eqs-empty by blast
qed
qed

```

```

then show ((Susp pi X, Paar t1 t2) # xs, ys) ∈ stuck
  unfolding stuck-def by simp
next
  case (fail-fresh-atom a ys)
  have ¬ (∃ P2 nabla s. ([], (a, Atom a) # ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ([], (a, Atom a) # ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-diff-atoms a b xs ys)
  hence ¬ (∃ P2 nabla s. ((Atom a, Atom b) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Atom a, Atom b) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-abst-unit a t xs ys)
  hence ¬ (∃ P2 nabla s. ((Abst a t, Unit) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Abst a t, Unit) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-abst-atom a t b xs ys)
  hence ¬ (∃ P2 nabla s. ((Abst a t, Atom b) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Abst a t, Atom b) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-abst-paar a t t1 t2 xs ys)
  hence ¬ (∃ P2 nabla s. ((Abst a t, Paar t1 t2) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Abst a t, Paar t1 t2) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-func-abst F t1 a t xs ys)
  hence ¬ (∃ P2 nabla s. ((Func F t1, Abst a t) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Func F t1, Abst a t) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-atom-unit b xs ys)
  hence ¬ (∃ P2 nabla s. ((Atom b, Unit) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Atom b, Unit) # xs, ys) ∈ stuck
    unfolding stuck-def by simp
next
  case (fail-paar-unit t1 t2 xs ys)
  hence ¬ (∃ P2 nabla s. ((Paar t1 t2, Unit) # xs, ys) ⊨ (nabla,s) ⇒ P2)
    by (auto elim: red-plus.cases s-red.cases c-red.cases)
  then show ((Paar t1 t2, Unit) # xs, ys) ∈ stuck

```

```

    unfolding stuck-def by simp
next
case (fail-func-unit F t1 xs ys)
hence  $\neg (\exists P2 \text{ nabra } s. ((\text{Func } F \ t1, \text{Unit}) \# \text{xs}, \text{ys}) \models (\text{nabra}, s) \Rightarrow P2)$ 
  by (auto elim: red-plus.cases s-red.cases c-red.cases)
then show  $((\text{Func } F \ t1, \text{Unit}) \# \text{xs}, \text{ys}) \in \text{stuck}$ 
  unfolding stuck-def by simp
next
case (fail-atom-paar a t1 t2 xs ys)
hence  $\neg (\exists P2 \text{ nabra } s. ((\text{Atom } a, \text{Paar } t1 \ t2) \# \text{xs}, \text{ys}) \models (\text{nabra}, s) \Rightarrow P2)$ 
  by (auto elim: red-plus.cases s-red.cases c-red.cases)
then show  $((\text{Atom } a, \text{Paar } t1 \ t2) \# \text{xs}, \text{ys}) \in \text{stuck}$ 
  unfolding stuck-def by simp
next
case (fail-func-atom F t1 a xs ys)
hence  $\neg (\exists P2 \text{ nabra } s. ((\text{Func } F \ t1, \text{Atom } a) \# \text{xs}, \text{ys}) \models (\text{nabra}, s) \Rightarrow P2)$ 
  by (auto elim: red-plus.cases s-red.cases c-red.cases)
then show  $((\text{Func } F \ t1, \text{Atom } a) \# \text{xs}, \text{ys}) \in \text{stuck}$ 
  unfolding stuck-def by simp
next
case (fail-func-paar F t t1 t2 xs ys)
hence  $\neg (\exists P2 \text{ nabra } s. ((\text{Func } F \ t, \text{Paar } t1 \ t2) \# \text{xs}, \text{ys}) \models (\text{nabra}, s) \Rightarrow P2)$ 
  by (auto elim: red-plus.cases s-red.cases c-red.cases)
then show  $((\text{Func } F \ t, \text{Paar } t1 \ t2) \# \text{xs}, \text{ys}) \in \text{stuck}$ 
  unfolding stuck-def by simp
next
case (fail-diff-func F1 F2 t1 t2 xs ys)
hence  $\neg (\exists P2 \text{ nabra } s. ((\text{Func } F1 \ t1, \text{Func } F2 \ t2) \# \text{xs}, \text{ys}) \models (\text{nabra}, s) \Rightarrow P2)$ 
  by (auto elim: red-plus.cases s-red.cases c-red.cases)
then show  $((\text{Func } F1 \ t1, \text{Func } F2 \ t2) \# \text{xs}, \text{ys}) \in \text{stuck}$ 
  unfolding stuck-def by simp
next
case (fail-sym s t xs ys)
hence not-reduce:  $\nexists P2 \text{ nabra } s1. ((s, t) \# \text{xs}, \text{ys}) \models (\text{nabra}, s1) \Rightarrow P2$ 
  unfolding stuck-def by simp
have  $\nexists P2 \text{ nabra } s1. ((t, s) \# \text{xs}, \text{ys}) \models (\text{nabra}, s1) \Rightarrow P2$ 
proof
  assume  $\exists P2 \text{ nabra } s1. ((t, s) \# \text{xs}, \text{ys}) \models (\text{nabra}, s1) \Rightarrow P2$ 
  then obtain  $P2 \text{ nabra } s1$  where
    reduces:  $((t, s) \# \text{xs}, \text{ys}) \models (\text{nabra}, s1) \Rightarrow P2$ 
    by auto
  hence  $\exists P3 \text{ nabra1 } s2. ((s, t) \# \text{xs}, \text{ys}) \models (\text{nabra1}, s2) \Rightarrow P3$ 
    using red-plus-symm[of  $\langle (t, s) \# \text{xs}, \text{ys} \rangle$   $t \ s \ \text{xs} \ \text{ys} \ \text{nabra } s1 \ P2$ ] by auto
  with not-reduce show False by simp
qed
then show  $((t, s) \# \text{xs}, \text{ys}) \in \text{stuck}$  unfolding stuck-def by simp
qed

```

```

lemma stuck-equiv:
  shows  $stuck = \{(\[], \[])\} \cup \{P1. fail P1\}$ 
proof (rule set-eqI, rule iffI)
  fix  $P$ 
  {assume  $P\text{-is-stuck}: P \in stuck$ 
  then obtain  $eqs\ freshs$  where
     $P\text{-def}: P = (eqs, freshs)$  by (cases P)
  show  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$ 
  proof(cases eqs)
    case  $Nil$ 
    then show  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$ 
    proof(cases freshs)
      case  $Nil$ 
      with  $\langle eqs = \[] \rangle$ 
      show  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$  using  $P\text{-def}$  by simp
    next
      case ( $Cons\ c\ freshs'$ )
      then obtain  $a\ t$  where  $c\text{-def}: c = a \#? t$  by force
      have  $t = Atom\ a$ 
      using fresh-reduces-if-not-atom P-is-stuck
      unfolding  $stuck\text{-def}\ P\text{-def}\ Nil\ Cons\ c\text{-def}$  by blast
      hence  $fail\ P$ 
      unfolding  $P\text{-def}\ Nil\ Cons\ c\text{-def}$  using fail-fresh-atom by simp
      thus  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$  by auto
    qed
  next
    case( $Cons\ e\ eqs'$ )
    then obtain  $s\ t$  where  $e\text{-def}: e = s \approx? t$  by force
    have  $fail\ P$ 
    using  $P\text{-is-stuck}$  unfolding  $P\text{-def}\ Cons\ e\text{-def}$ 
       $stuck\text{-def}$  using not-reduce-then-fail by simp
    thus  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$  by auto
    qed }

  {assume  $P \in \{(\[], \[])\} \cup \{P1. fail P1\}$ 
  then show  $P \in stuck$ 
    using empty-stuck fail-is-stuck by blast
  }
qed

```

```

lemma u-empty-sred:
  assumes  $P1 \vdash s \rightsquigarrow P2$  and  $U\ P2 = \{\}$ 
  shows  $U\ P1 = \{\}$ 
  using assms P1-from-P2-sred all-solutions-def P1-to-P2-sred by blast

```

**lemma** *u-empty-cred*:  
**assumes**  $P1 \vdash \text{nabla} \rightarrow P2$  **and**  $U P2 = \{\}$   
**shows**  $U P1 = \{\}$   
**using** *assms P1-from-P2-cred all-solutions-def P1-to-P2-cred* **by** *blast*

**lemma** *u-empty-red-plus*:  
**assumes**  $P1 \models (\text{nabla}, s) \Rightarrow P2$  **and**  $U P2 = \{\}$   
**shows**  $U P1 = \{\}$   
**using** *assms P1-from-P2-red-plus all-solutions-def P1-to-P2-red-plus1* **by** *fast*

**lemma** *empty-then-fail*:  
**assumes**  $U P1 = \{\}$   
**shows**  $(\forall P \in \text{normal-form } P1. \text{fail } P)$   
**proof**  
**fix**  $P$   
**assume**  $P\text{-is-nf}: P \in \text{normal-form } P1$   
**hence**  $P\text{-is-stuck}: P \in \text{stuck}$   
**using** *normal-form-def* **by**  $(\text{cases } P1 \in \text{stuck}) \text{ auto}$   
**hence**  $P\text{-is-empty-or-fails}: P = ([], []) \vee \text{fail } P$   
**using** *stuck-equiv* **by** *auto*  
**have**  $P \neq ([], [])$   
**proof**  
**assume**  $P\text{-empty}: P = ([], [])$   
**hence**  $\text{solution}: (\{\}, []) \in U P$   
**using** *all-solutions-def* **by** *simp*  
**hence**  $P\text{-neq}: P \neq P1$   
**using** *assms* **by** *auto*  
**show** *False*  
**proof**  $(\text{cases } P1 \in \text{stuck})$   
**case** *True*  
**then** **have**  $\text{normal-form } P1 = \{P1\}$   
**unfolding** *normal-form-def* **by** *simp*  
**with**  $P\text{-is-nf}$  **have**  $P = P1$  **by** *simp*  
**with**  $P\text{-neq}$  **show** *False* **by** *simp*  
**next**  
**case** *False*  
**with**  $P\text{-is-nf}$   
**obtain**  $\text{nabla } s$  **where**  $P1\text{-goes-to-}P: P1 \models (\text{nabla}, s) \Rightarrow P$   
**unfolding** *normal-form-def* **by** *auto*  
**hence**  $(\{\} \cup \text{nabla}, [] \cdot s) \in U P1$   
**using**  $P1\text{-from-}P2\text{-red-plus}[OF P1\text{-goes-to-}P \text{ solution ext-subst-id}]$  **by** *simp*  
**with** *assms* **show** *False* **by** *simp*  
**qed**  
**qed**  
**thus** *fail P*

using *P-is-empty-or-fails* by *simp*  
 qed

**lemma** *not-empty-then-not-fail*:  
 assumes  $U\ P1 \neq \{\}$   
 shows  $\neg(\exists P \in \text{normal-form } P1. \text{fail } P)$   
**proof**(*simp*, rule *ballI*)  
 fix  $P$   
 assume *P-is-nf*:  $P \in \text{normal-form } P1$   
 show  $\neg \text{fail } P$   
**proof**  
 assume *P-fails*: *fail P*  
 show *False*  
**proof**(cases  $P1 \in \text{stuck}$ )  
 case *True*  
 have *normal-form*  $P1 = \{P1\}$   
 unfolding *normal-form-def* using *True* by *simp*  
 hence  $P = P1$  using *P-is-nf* by *simp*  
 with *P-fails* have  $U\ P1 = \{\}$   
 using *fail-then-empty* by *simp*  
 thus *False* using *assms* by *simp*  
 next  
 case *False*  
 with *P-is-nf*  
 obtain *nabla s* where *P1-goes-to-P*:  $P1 \models (nabla, s) \Rightarrow P$   
 unfolding *normal-form-def* by *auto*  
 moreover have  $U\ P = \{\}$   
 using *P-fails fail-then-empty* by *simp*  
 ultimately have  $U\ P1 = \{\}$   
 using *u-empty-red-plus* by *simp*  
 then show *False*  
 using *assms* by *simp*  
 qed  
 qed  
 qed

## References

- [1] M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, D. Nantes-Sobrinho, and A. C. R. Oliveira. A formalisation of nominal  $\alpha$ -

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.