

# Verified Metatheory and Type Inference for a Name-Carrying Simply-Typed $\lambda$ -Calculus

Michael Rawson

June 11, 2019

## Abstract

I formalise a Church-style simply-typed  $\lambda$ -calculus, extended with pairs, a unit value, and projection functions, and show some metatheory of the calculus, such as the subject reduction property. Particular attention is paid to the treatment of names in the calculus. A nominal style of binding is used, but I use a manual approach over Nominal Isabelle in order to extract an executable type inference algorithm. More information can be found in my [undergraduate dissertation](#).

## Contents

```
theory Fresh
imports Main
begin
```

```
class fresh =
  fixes fresh-in :: 'a set  $\Rightarrow$  'a
  assumes finite S  $\implies$  fresh-in S  $\notin$  S
```

```
instantiation nat :: fresh
```

```
begin
```

```
definition fresh-in-nat :: nat set  $\Rightarrow$  nat where
  [code]: fresh-in-nat S  $\equiv$  (if Set.is-empty S then 0 else Max S + 1)
```

```
instance proof
```

```
fix S :: nat set
```

```
assume finite S
```

```
consider Set.is-empty S |  $\neg$ Set.is-empty S by auto
```

```
thus fresh-in S  $\notin$  S unfolding fresh-in-nat-def
```

```
proof(cases)
```

```
case 1
```

```
hence S = {} using Set.is-empty-def by metis
```

```
hence 0  $\notin$  S by auto
```

```
thus (if Set.is-empty S then 0 else Max S + 1)  $\notin$  S using 1 by auto
```

```
next
```

```

    case 2
      have  $Max\ S + 1 \notin S$ 
        using ⟨finite S⟩ Max.coboundedI add-le-same-cancel1 not-one-le-zero
        by blast
      thus (if Set.is-empty S then 0 else  $Max\ S + 1$ )  $\notin S$  using 2 by auto
    next
  qed
qed
end

end
theory Permutation
imports Main
begin

type-synonym 'a swp = 'a × 'a
type-synonym 'a preprm = 'a swp list

definition preprm-id :: 'a preprm where preprm-id = []

fun swp-apply :: 'a swp ⇒ 'a ⇒ 'a where
  swp-apply (a, b) x = (if x = a then b else (if x = b then a else x))

fun preprm-apply :: 'a preprm ⇒ 'a ⇒ 'a where
  preprm-apply [] x = x
| preprm-apply (s # ss) x = swp-apply s (preprm-apply ss x)

definition preprm-compose :: 'a preprm ⇒ 'a preprm ⇒ 'a preprm where
  preprm-compose f g ≡ f @ g

definition preprm-unit :: 'a ⇒ 'a ⇒ 'a preprm where
  preprm-unit a b ≡ [(a, b)]

definition preprm-ext :: 'a preprm ⇒ 'a preprm ⇒ bool (infix =p 100) where
  π =p σ ≡ ∀ x. preprm-apply π x = preprm-apply σ x

definition preprm-inv :: 'a preprm ⇒ 'a preprm where
  preprm-inv π ≡ rev π

lemma swp-apply-unequal:
  assumes  $x \neq y$ 
  shows  $swp-apply\ s\ x \neq swp-apply\ s\ y$ 
proof (cases s)
  case (Pair a b)
  consider  $x = a \mid x = b \mid x \neq a \wedge x \neq b$  by auto
  thus ?thesis proof (cases)
    case 1
    have  $swp-apply\ s\ x = b$  using ⟨s = (a, b)⟩ ⟨x = a⟩ by simp
    moreover have  $swp-apply\ s\ y \neq b$  using ⟨s = (a, b)⟩ ⟨x = a⟩ ⟨x ≠ y⟩

```

```

    by(cases y = b, simp-all)
  ultimately show ?thesis by metis
next
case 2
  have swp-apply s x = a using ⟨s = (a, b)⟩ ⟨x = b⟩ by simp
  moreover have swp-apply s y ≠ a using ⟨s = (a, b)⟩ ⟨x = b⟩ ⟨x ≠ y⟩
    by(cases y = a, simp-all)
  ultimately show ?thesis by metis
next
case 3
  have swp-apply s x = x using ⟨s = (a, b)⟩ ⟨x ≠ a ∧ x ≠ b⟩ by simp
  consider y = a | y = b | y ≠ a ∧ y ≠ b by auto
  hence swp-apply s y ≠ x proof(cases)
    case 1
      hence swp-apply s y = b using ⟨s = (a, b)⟩ by simp
      thus ?thesis using ⟨x ≠ a ∧ x ≠ b⟩ by metis
    next
    case 2
      hence swp-apply s y = a using ⟨s = (a, b)⟩ by simp
      thus ?thesis using ⟨x ≠ a ∧ x ≠ b⟩ by metis
    next
    case 3
      hence swp-apply s y = y using ⟨s = (a, b)⟩ by simp
      thus ?thesis using ⟨x ≠ y⟩ by metis
  next
  qed
  thus ?thesis using ⟨swp-apply s x = x⟩ ⟨x ≠ y⟩ by metis
next
qed
next
qed

```

**lemma** *preprm-ext-reflexive*:  
 shows  $x =_p x$   
 unfolding *preprm-ext-def* by auto

**corollary** *preprm-ext-reflp*:  
 shows *reflp preprm-ext*  
 unfolding *reflp-def* using *preprm-ext-reflexive* by auto

**lemma** *preprm-ext-symmetric*:  
 assumes  $x =_p y$   
 shows  $y =_p x$   
 using *assms* unfolding *preprm-ext-def* by auto

**corollary** *preprm-ext-symp*:  
 shows *symp preprm-ext*  
 unfolding *symp-def* using *preprm-ext-symmetric* by auto

**lemma** *preprm-ext-transitive*:  
 assumes  $x =_p y$  and  $y =_p z$   
 shows  $x =_p z$   
**using** *assms unfolding preprm-ext-def by auto*

**corollary** *preprm-ext-transp*:  
 shows *transp preprm-ext*  
**unfolding** *transp-def using preprm-ext-transitive by auto*

**lemma** *preprm-apply-composition*:  
 shows *preprm-apply (preprm-compose f g) x = preprm-apply f (preprm-apply g x)*  
**unfolding** *preprm-compose-def*  
**by** (*induction f, simp-all*)

**lemma** *preprm-apply-unequal*:  
 assumes  $x \neq y$   
 shows *preprm-apply  $\pi$  x  $\neq$  preprm-apply  $\pi$  y*  
**using** *assms proof (induction  $\pi$ , simp)*  
**case** (*Cons s ss*)  
 have *preprm-apply (s # ss) x = swp-apply s (preprm-apply ss x)*  
 and *preprm-apply (s # ss) y = swp-apply s (preprm-apply ss y)* **by** *auto*  
**thus** *?case using Cons.IH  $\langle x \neq y \rangle$  swp-apply-unequal by metis*  
**next**  
**qed**

**lemma** *preprm-unit-equal-id*:  
 shows *preprm-unit a a =<sub>p</sub> preprm-id*  
**unfolding** *preprm-ext-def preprm-unit-def preprm-id-def*  
**by** *simp*

**lemma** *preprm-unit-inaction*:  
 assumes  $x \neq a$  and  $x \neq b$   
 shows *preprm-apply (preprm-unit a b) x = x*  
**unfolding** *preprm-unit-def using assms by simp*

**lemma** *preprm-unit-action*:  
 shows *preprm-apply (preprm-unit a b) a = b*  
**unfolding** *preprm-unit-def by simp*

**lemma** *preprm-unit-commutes*:  
 shows *preprm-unit a b =<sub>p</sub> preprm-unit b a*  
**unfolding** *preprm-ext-def preprm-unit-def*  
**by** *simp*

**lemma** *preprm-singleton-involution*:  
 shows *preprm-compose [s] [s] =<sub>p</sub> preprm-id*  
**unfolding** *preprm-ext-def preprm-compose-def preprm-unit-def preprm-id-def*  
**proof** –

**obtain**  $s1\ s2$  **where**  $s1 = fst\ s\ s2 = snd\ s$  **by** *auto*  
**hence**  $s = (s1, s2)$  **by** *simp*  
**thus**  $\forall x. preprm-apply\ ([s]\ @\ [s])\ x = preprm-apply\ []\ x$   
**by** *simp*  
**qed**

**lemma** *preprm-unit-involution*:  
**shows**  $preprm-compose\ (preprm-unit\ a\ b)\ (preprm-unit\ a\ b) =_p\ preprm-id$   
**unfolding** *preprm-unit-def*  
**using** *preprm-singleton-involution*.

**lemma** *preprm-apply-id*:  
**shows**  $preprm-apply\ preprm-id\ x = x$   
**unfolding** *preprm-id-def*  
**by** *simp*

**lemma** *preprm-apply-injective*:  
**shows**  $inj\ (preprm-apply\ \pi)$   
**unfolding** *inj-on-def* **proof**(*rule+*)  
**fix**  $x\ y$   
**assume**  $preprm-apply\ \pi\ x = preprm-apply\ \pi\ y$   
**thus**  $x = y$  **proof**(*induction\ \pi*)  
**case** *Nil*  
**thus** *?case* **by** *auto*  
**next**  
**case** (*Cons\ s\ ss*)  
**hence**  $swp-apply\ s\ (preprm-apply\ ss\ x) = swp-apply\ s\ (preprm-apply\ ss\ y)$  **by**  
*auto*  
**thus** *?case* **using** *swp-apply-unequal\ Cons.IH* **by** *metis*  
**next**  
**qed**  
**qed**

**lemma** *preprm-disagreement-composition*:  
**assumes**  $a \neq b\ b \neq c\ a \neq c$   
**shows**  $\{x. preprm-apply\ (preprm-compose\ (preprm-unit\ a\ b)\ (preprm-unit\ b\ c))\ x \neq preprm-apply\ (preprm-unit\ a\ c)\ x\} = \{a, b\}$   
**unfolding** *preprm-unit-def\ preprm-compose-def* **proof**  
**show**  $\{x. preprm-apply\ ([a, b])\ @\ [(b, c)]\ x \neq preprm-apply\ [(a, c)]\ x\} \subseteq \{a, b\}$   
**proof**  
**fix**  $x$   
**have**  $x \notin \{a, b\} \implies x \notin \{x. preprm-apply\ ([a, b])\ @\ [(b, c)]\ x \neq preprm-apply\ [(a, c)]\ x\}$   
**proof** –  
**assume**  $x \notin \{a, b\}$   
**hence**  $x \neq a \wedge x \neq b$  **by** *auto*

**hence**  $\text{preprm-apply } [(a, b)] @ [(b, c)] x = \text{preprm-apply } [(a, c)] x$  **by** *simp*  
**thus**  $x \notin \{x. \text{preprm-apply } [(a, b)] @ [(b, c)] x \neq \text{preprm-apply } [(a, c)] x\}$   
**by** *auto*  
**qed**  
**thus**  $x \in \{x. \text{preprm-apply } [(a, b)] @ [(b, c)] x \neq \text{preprm-apply } [(a, c)] x\}$   
 $\implies x \in \{a, b\}$   
**by** *blast*  
**qed**  
**show**  $\{a, b\} \subseteq \{x. \text{preprm-apply } [(a, b)] @ [(b, c)] x \neq \text{preprm-apply } [(a, c)] x\}$   
**proof**  
**fix**  $x$   
**assume**  $x \in \{a, b\}$   
**from** *this* **consider**  $x = a \mid x = b$  **by** *auto*  
**thus**  $x \in \{x. \text{preprm-apply } [(a, b)] @ [(b, c)] x \neq \text{preprm-apply } [(a, c)] x\}$   
**using** *assms* **by** (*cases, simp-all*)  
**qed**  
**qed**

**lemma** *preprm-compose-push*:

**shows**

$\text{preprm-compose } \pi (\text{preprm-unit } a b) = p$

$\text{preprm-compose } (\text{preprm-unit } (\text{preprm-apply } \pi a) (\text{preprm-apply } \pi b)) \pi$

**unfolding** *preprm-ext-def preprm-unit-def*

**by** (*simp add: inj-eq preprm-apply-composition preprm-apply-injective*)

**lemma** *preprm-ext-compose-left*:

**assumes**  $P =_p S$

**shows**  $\text{preprm-compose } \pi P =_p \text{preprm-compose } \pi S$

**using** *assms* **unfolding** *preprm-ext-def*

**using** *preprm-apply-composition* **by** *metis*

**lemma** *preprm-ext-compose-right*:

**assumes**  $P =_p S$

**shows**  $\text{preprm-compose } P \pi =_p \text{preprm-compose } S \pi$

**using** *assms* **unfolding** *preprm-ext-def*

**using** *preprm-apply-composition* **by** *metis*

**lemma** *preprm-ext-uncompose*:

**assumes**  $\pi =_p \sigma$   $\text{preprm-compose } \pi P =_p \text{preprm-compose } \sigma S$

**shows**  $P =_p S$

**using** *assms* **unfolding** *preprm-ext-def*

**proof** –

**assume**  $*$ :  $\forall x. \text{preprm-apply } \pi x = \text{preprm-apply } \sigma x$

**assume**  $\forall x. \text{preprm-apply } (\text{preprm-compose } \pi P) x = \text{preprm-apply } (\text{preprm-compose } \sigma S) x$

**hence**  $\forall x. \text{preprm-apply } \pi (\text{preprm-apply } P x) = \text{preprm-apply } \sigma (\text{preprm-apply } S x)$

$S x$ )  
**using** *preprm-apply-composition* **by** *metis*  
**hence**  $\forall x. \text{preprm-apply } \pi (\text{preprm-apply } P x) = \text{preprm-apply } \pi (\text{preprm-apply } S x)$   
 $S x$ )  
**using**  $*$  **by** *metis*  
**thus**  $\forall x. \text{preprm-apply } P x = \text{preprm-apply } S x$   
**using** *preprm-apply-injective* **unfolding** *inj-on-def* **by** *fastforce*  
**qed**

**lemma** *preprm-inv-compose*:  
**shows**  $\text{preprm-compose } (\text{preprm-inv } \pi) \pi =_p \text{preprm-id}$   
**unfolding** *preprm-inv-def*  
**proof**(*induction*  $\pi$ , *simp* *add*: *preprm-ext-def* *preprm-id-def* *preprm-compose-def*)  
**case** (*Cons*  $p$   $ps$ )  
**hence** *IH*:  $(\text{preprm-compose } (\text{rev } ps) ps) =_p \text{preprm-id}$  **by** *auto*  
  
**have**  $(\text{preprm-compose } (\text{rev } (p \# ps)) (p \# ps)) =_p (\text{preprm-compose } (\text{rev } ps)$   
 $(\text{preprm-compose } (\text{preprm-compose } [p] [p]) ps))$   
**unfolding** *preprm-compose-def* **using** *preprm-ext-reflexive* **by** *simp*  
**moreover** **have**  $\dots =_p (\text{preprm-compose } (\text{rev } ps) (\text{preprm-compose } \text{preprm-id}$   
 $ps))$   
**using** *preprm-singleton-involution* *preprm-ext-compose-left* *preprm-ext-compose-right*  
**by** *metis*  
**moreover** **have**  $\dots =_p (\text{preprm-compose } (\text{rev } ps) ps)$   
**unfolding** *preprm-compose-def* *preprm-id-def* **using** *preprm-ext-reflexive* **by**  
*simp*  
**moreover** **have**  $\dots =_p \text{preprm-id}$  **using** *IH*.  
**ultimately show** *?case* **using** *preprm-ext-transitive* **by** *metis*  
**next**  
**qed**

**lemma** *preprm-inv-involution*:  
**shows**  $\text{preprm-inv } (\text{preprm-inv } \pi) = \pi$   
**unfolding** *preprm-inv-def* **by** *simp*

**lemma** *preprm-inv-ext*:  
**assumes**  $\pi =_p \sigma$   
**shows**  $\text{preprm-inv } \pi =_p \text{preprm-inv } \sigma$   
**proof** –  
**have**  
 $(\text{preprm-compose } (\text{preprm-inv } (\text{preprm-inv } \pi)) (\text{preprm-inv } \pi)) =_p \text{preprm-id}$   
 $(\text{preprm-compose } (\text{preprm-inv } (\text{preprm-inv } \sigma)) (\text{preprm-inv } \sigma)) =_p \text{preprm-id}$   
**using** *preprm-inv-compose* **by** *metis+*  
**hence**  
 $(\text{preprm-compose } \pi (\text{preprm-inv } \pi)) =_p \text{preprm-id}$   
 $(\text{preprm-compose } \sigma (\text{preprm-inv } \sigma)) =_p \text{preprm-id}$   
**using** *preprm-inv-involution* **by** *metis+*  
**hence**  $(\text{preprm-compose } \pi (\text{preprm-inv } \pi)) =_p (\text{preprm-compose } \sigma (\text{preprm-inv } \sigma))$   
 $\sigma))$

**using** *preprm-ext-transitive preprm-ext-symmetric* **by** *metis*  
**thus** *preprm-inv  $\pi = p$  preprm-inv  $\sigma$*   
**using** *preprm-ext-uncompose assms* **by** *metis*  
**qed**

**quotient-type** *'a prm = 'a preprm / preprm-ext*  
**proof**(*rule equivpI*)  
**show** *reflp preprm-ext* **using** *preprm-ext-reflp*.  
**show** *symp preprm-ext* **using** *preprm-ext-symp*.  
**show** *transp preprm-ext* **using** *preprm-ext-transp*.  
**qed**

**lift-definition** *prm-id :: 'a prm ( $\varepsilon$ ) is preprm-id.*

**lift-definition** *prm-apply :: 'a prm  $\Rightarrow$  'a  $\Rightarrow$  'a (infix \$ 140) is preprm-apply*  
**unfolding** *preprm-ext-def*  
**using** *preprm-apply.simps* **by** *auto*

**lift-definition** *prm-compose :: 'a prm  $\Rightarrow$  'a prm  $\Rightarrow$  'a prm (infixr  $\diamond$  145) is*  
*preprm-compose*  
**unfolding** *preprm-ext-def*  
**by**(*simp only: preprm-apply-composition, simp*)

**lift-definition** *prm-unit :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a prm ( $[- \leftrightarrow -]$ ) is preprm-unit.*

**lift-definition** *prm-inv :: 'a prm  $\Rightarrow$  'a prm is preprm-inv*  
**using** *preprm-inv-ext*.

**lemma** *prm-apply-composition:*  
**fixes** *f g :: 'a prm and x :: 'a*  
**shows** *f  $\diamond$  g \$ x = f \$ (g \$ x)*  
**by**(*transfer, metis preprm-apply-composition*)

**lemma** *prm-apply-unequal:*  
**fixes**  *$\pi :: 'a prm$  and  $x y :: 'a$*   
**assumes**  *$x \neq y$*   
**shows**  *$\pi $ x \neq \pi $ y$*   
**using** *assms* **by** (*transfer, metis preprm-apply-unequal*)

**lemma** *prm-unit-equal-id:*  
**fixes** *a :: 'a*  
**shows**  *$[a \leftrightarrow a] = \varepsilon$*   
**by** (*transfer, metis preprm-unit-equal-id*)

**lemma** *prm-unit-inaction:*  
**fixes** *a b x :: 'a*  
**assumes**  *$x \neq a$  and  $x \neq b$*   
**shows**  *$[a \leftrightarrow b] $ x = x$*   
**using** *assms*



by (transfer, metis preprm-unit-inaction)

**lemma** *prm-unit-action*:

fixes  $a b :: 'a$

shows  $[a \leftrightarrow b] \$ a = b$

by (transfer, metis preprm-unit-action)

**lemma** *prm-unit-commutes*:

fixes  $a b :: 'a$

shows  $[a \leftrightarrow b] = [b \leftrightarrow a]$

by (transfer, metis preprm-unit-commutes)

**lemma** *prm-unit-involution*:

fixes  $a b :: 'a$

shows  $[a \leftrightarrow b] \diamond [a \leftrightarrow b] = \varepsilon$

by (transfer, metis preprm-unit-involution)

**lemma** *prm-apply-id*:

fixes  $x :: 'a$

shows  $\varepsilon \$ x = x$

by (transfer, metis preprm-apply-id)

**lemma** *prm-apply-injective*:

shows inj (prm-apply  $\pi$ )

by (transfer, metis preprm-apply-injective)

**lemma** *prm-inv-compose*:

shows (prm-inv  $\pi$ )  $\diamond \pi = \varepsilon$

by (transfer, metis preprm-inv-compose)

**interpretation**  $'a$  prm: semigroup prm-compose

**unfolding** semigroup-def **by** (transfer, simp add: preprm-compose-def preprm-ext-def)

**interpretation**  $'a$  prm: group prm-compose prm-id prm-inv

**unfolding** group-def group-axioms-def

**proof** –

have semigroup ( $\diamond$ ) **using**  $'a$  prm.semigroup-axioms.

**moreover have**  $\forall a. \varepsilon \diamond a = a$  **by** (transfer, simp add: preprm-id-def preprm-compose-def preprm-ext-def)

**moreover have**  $\forall a. \text{prm-inv } a \diamond a = \varepsilon$  **using** prm-inv-compose **by** blast

**ultimately show** semigroup ( $\diamond$ )  $\wedge (\forall a. \varepsilon \diamond a = a) \wedge (\forall a. \text{prm-inv } a \diamond a = \varepsilon)$

**by** blast

**qed**

**definition** *prm-set* ::  $'a$  prm  $\Rightarrow$   $'a$  set  $\Rightarrow$   $'a$  set (**infix**  $\{\$\}$  140) **where**

$\text{prm-set } \pi S \equiv \text{image } (\text{prm-apply } \pi) S$

**lemma** *prm-set-apply-compose*:

shows  $\pi \{\$\} (\sigma \{\$\} S) = (\pi \diamond \sigma) \{\$\} S$

**unfolding** *prm-set-def* **proof** –  
**have**  $(\$)\ \pi \text{ ' } (\$)\ \sigma \text{ ' } S = (\lambda x. \pi \$ x) \text{ ' } (\lambda x. \sigma \$ x) \text{ ' } S$  **by** *simp*  
**moreover have**  $\dots = (\lambda x. \pi \$ (\sigma \$ x)) \text{ ' } S$  **by** *auto*  
**moreover have**  $\dots = (\lambda x. (\pi \diamond \sigma) \$ x) \text{ ' } S$  **using** *prm-apply-composition* **by**  
*metis*  
**moreover have**  $\dots = (\pi \diamond \sigma) \{ \$ \} S$  **using** *prm-set-def* **by** *metis*  
**ultimately show**  $(\$)\ \pi \text{ ' } (\$)\ \sigma \text{ ' } S = (\$)\ (\pi \diamond \sigma) \text{ ' } S$  **by** *metis*  
**qed**

**lemma** *prm-set-membership*:  
**assumes**  $x \in S$   
**shows**  $\pi \$ x \in \pi \{ \$ \} S$   
**using** *assms* **unfolding** *prm-set-def* **by** *simp*

**lemma** *prm-set-notmembership*:  
**assumes**  $x \notin S$   
**shows**  $\pi \$ x \notin \pi \{ \$ \} S$   
**using** *assms* **unfolding** *prm-set-def*  
**by** (*simp add: inj-image-mem-iff prm-apply-injective*)

**lemma** *prm-set-singleton*:  
**shows**  $\pi \{ \$ \} \{ x \} = \{ \pi \$ x \}$   
**unfolding** *prm-set-def* **by** *auto*

**lemma** *prm-set-id*:  
**shows**  $\varepsilon \{ \$ \} S = S$   
**unfolding** *prm-set-def*  
**proof** –  
**have**  $(\$)\ \varepsilon \text{ ' } S = (\lambda x. \varepsilon \$ x) \text{ ' } S$  **by** *simp*  
**moreover have**  $\dots = (\lambda x. x) \text{ ' } S$  **using** *prm-apply-id* **by** *metis*  
**moreover have**  $\dots = S$  **by** *auto*  
**ultimately show**  $(\$)\ \varepsilon \text{ ' } S = S$  **by** *metis*  
**qed**

**lemma** *prm-set-unit-inaction*:  
**assumes**  $a \notin S$  **and**  $b \notin S$   
**shows**  $[a \leftrightarrow b] \{ \$ \} S = S$   
**proof**  
**show**  $[a \leftrightarrow b] \{ \$ \} S \subseteq S$  **proof**  
**fix**  $x$   
**assume**  $H: x \in [a \leftrightarrow b] \{ \$ \} S$   
**from this obtain**  $y$  **where**  $x = [a \leftrightarrow b] \$ y$  **unfolding** *prm-set-def* **using**  
*imageE* **by** *metis*  
**hence**  $y \in S$  **using**  $H$  *inj-image-mem-iff* *prm-apply-injective* *prm-set-def* **by**  
*metis*  
**hence**  $y \neq a$  **and**  $y \neq b$  **using** *assms* **by** *auto*  
**hence**  $x = y$  **using** *prm-unit-inaction*  $\langle x = [a \leftrightarrow b] \$ y \rangle$  **by** *metis*  
**thus**  $x \in S$  **using**  $\langle y \in S \rangle$  **by** *auto*  
**qed**

**show**  $S \subseteq [a \leftrightarrow b] \{\$\} S$  **proof**  
**fix**  $x$   
**assume**  $H: x \in S$   
**hence**  $x \neq a$  **and**  $x \neq b$  **using** *assms* **by** *auto*  
**hence**  $x = [a \leftrightarrow b] \{\$\} x$  **using** *prm-unit-inaction* **by** *metis*  
**thus**  $x \in [a \leftrightarrow b] \{\$\} S$  **unfolding** *prm-set-def* **using**  $H$  **by** *simp*  
**qed**  
**qed**

**lemma** *prm-set-unit-action*:  
**assumes**  $a \in S$  **and**  $b \notin S$   
**shows**  $[a \leftrightarrow b] \{\$\} S = S - \{a\} \cup \{b\}$   
**proof**  
**show**  $[a \leftrightarrow b] \{\$\} S \subseteq S - \{a\} \cup \{b\}$  **proof**  
**fix**  $x$   
**assume**  $H: x \in [a \leftrightarrow b] \{\$\} S$   
**from** *this* **obtain**  $y$  **where**  $x = [a \leftrightarrow b] \{\$\} y$  **unfolding** *prm-set-def* **using** *imageE* **by** *metis*  
**hence**  $y \in S$  **using**  $H$  *inj-image-mem-iff* *prm-apply-injective* *prm-set-def* **by** *metis*  
**hence**  $y \neq b$  **using** *assms* **by** *auto*  
**consider**  $y = a \mid y \neq a$  **by** *auto*  
**thus**  $x \in S - \{a\} \cup \{b\}$  **proof**(*cases*)  
**case** 1  
**hence**  $x = b$  **using**  $\langle x = [a \leftrightarrow b] \{\$\} y \rangle$  **using** *prm-unit-action* **by** *metis*  
**thus** *?thesis* **by** *auto*  
**next**  
**case** 2  
**hence**  $x = y$  **using**  $\langle x = [a \leftrightarrow b] \{\$\} y \rangle$  **using** *prm-unit-inaction*  $\langle y \neq b \rangle$  **by** *metis*  
**hence**  $x \in S$  **and**  $x \neq a$  **using**  $\langle y \in S \rangle \langle y \neq a \rangle$  **by** *auto*  
**thus** *?thesis* **by** *auto*  
**next**  
**qed**  
**qed**

**show**  $S - \{a\} \cup \{b\} \subseteq [a \leftrightarrow b] \{\$\} S$  **proof**  
**fix**  $x$   
**assume**  $H: x \in S - \{a\} \cup \{b\}$   
**hence**  $x \neq a$  **using** *assms* **by** *auto*  
**consider**  $x = b \mid x \neq b$  **by** *auto*  
**thus**  $x \in [a \leftrightarrow b] \{\$\} S$  **proof**(*cases*)  
**case** 1  
**hence**  $x = [a \leftrightarrow b] \{\$\} a$  **using** *prm-unit-action* **by** *metis*  
**thus** *?thesis* **using**  $\langle a \in S \rangle$  *prm-set-membership* **by** *metis*  
**next**  
**case** 2  
**hence**  $x \in S$  **using**  $H$  **by** *auto*  
**moreover** **have**  $x = [a \leftrightarrow b] \{\$\} x$  **using** *prm-unit-inaction*  $\langle x \neq a \rangle \langle x \neq b \rangle$   
**by** *metis*

```

      ultimately show ?thesis using prm-set-membership by metis
    next
    qed
  qed
qed

lemma prm-set-distributes-union:
  shows  $\pi \{ \$ \} (S \cup T) = (\pi \{ \$ \} S) \cup (\pi \{ \$ \} T)$ 
unfolding prm-set-def by auto

lemma prm-set-distributes-difference:
  shows  $\pi \{ \$ \} (S - T) = (\pi \{ \$ \} S) - (\pi \{ \$ \} T)$ 
unfolding prm-set-def using prm-apply-injective image-set-diff by metis

definition prm-disagreement :: 'a prm  $\Rightarrow$  'a prm  $\Rightarrow$  'a set (ds) where
  prm-disagreement  $\pi \sigma \equiv \{x. \pi \$ x \neq \sigma \$ x\}$ 

lemma prm-disagreement-ext:
  shows  $x \in ds \pi \sigma \equiv \pi \$ x \neq \sigma \$ x$ 
unfolding prm-disagreement-def by simp

lemma prm-disagreement-composition:
  assumes  $a \neq b \ b \neq c \ a \neq c$ 
  shows  $ds ([a \leftrightarrow b] \diamond [b \leftrightarrow c]) [a \leftrightarrow c] = \{a, b\}$ 
using assms unfolding prm-disagreement-def by (transfer, metis preprm-disagreement-composition)

lemma prm-compose-push:
  shows  $\pi \diamond [a \leftrightarrow b] = [\pi \$ a \leftrightarrow \pi \$ b] \diamond \pi$ 
by (transfer, metis preprm-compose-push)

end
theory PreSimplyTyped
imports Fresh Permutation
begin

type-synonym tvar = nat

datatype type =
  TUnit
| TVar tvar
| TArr type type
| TPair type type

datatype 'a ptrm =
  PUnit
| PVar 'a
| PApp 'a ptrm 'a ptrm
| PFn 'a type 'a ptrm
| PPair 'a ptrm 'a ptrm

```

| *PFst* 'a *ptrm*  
| *PSnd* 'a *ptrm*

**fun** *ptrm-fvs* :: 'a *ptrm*  $\Rightarrow$  'a *set* **where**  
*ptrm-fvs* *PUnit* = {}  
| *ptrm-fvs* (*PVar* *x*) = {*x*}  
| *ptrm-fvs* (*PApp* *A* *B*) = *ptrm-fvs* *A*  $\cup$  *ptrm-fvs* *B*  
| *ptrm-fvs* (*PFn* *x* - *A*) = *ptrm-fvs* *A* - {*x*}  
| *ptrm-fvs* (*PPair* *A* *B*) = *ptrm-fvs* *A*  $\cup$  *ptrm-fvs* *B*  
| *ptrm-fvs* (*PFst* *P*) = *ptrm-fvs* *P*  
| *ptrm-fvs* (*PSnd* *P*) = *ptrm-fvs* *P*

**fun** *ptrm-apply-prm* :: 'a *prm*  $\Rightarrow$  'a *ptrm*  $\Rightarrow$  'a *ptrm* (**infixr** · 150) **where**  
*ptrm-apply-prm*  $\pi$  *PUnit* = *PUnit*  
| *ptrm-apply-prm*  $\pi$  (*PVar* *x*) = *PVar* ( $\pi$  \$ *x*)  
| *ptrm-apply-prm*  $\pi$  (*PApp* *A* *B*) = *PApp* (*ptrm-apply-prm*  $\pi$  *A*) (*ptrm-apply-prm*  $\pi$  *B*)  
| *ptrm-apply-prm*  $\pi$  (*PFn* *x* *T* *A*) = *PFn* ( $\pi$  \$ *x*) *T* (*ptrm-apply-prm*  $\pi$  *A*)  
| *ptrm-apply-prm*  $\pi$  (*PPair* *A* *B*) = *PPair* (*ptrm-apply-prm*  $\pi$  *A*) (*ptrm-apply-prm*  $\pi$  *B*)  
| *ptrm-apply-prm*  $\pi$  (*PFst* *P*) = *PFst* (*ptrm-apply-prm*  $\pi$  *P*)  
| *ptrm-apply-prm*  $\pi$  (*PSnd* *P*) = *PSnd* (*ptrm-apply-prm*  $\pi$  *P*)

**inductive** *ptrm-alpha-equiv* :: 'a *ptrm*  $\Rightarrow$  'a *ptrm*  $\Rightarrow$  *bool* (**infix**  $\approx$  100) **where**  
*unit*: *PUnit*  $\approx$  *PUnit*  
| *var*: (*PVar* *x*)  $\approx$  (*PVar* *x*)  
| *app*: [*A*  $\approx$  *B*; *C*  $\approx$  *D*]  $\Longrightarrow$  (*PApp* *A* *C*)  $\approx$  (*PApp* *B* *D*)  
| *fn1*: *A*  $\approx$  *B*  $\Longrightarrow$  (*PFn* *x* *T* *A*)  $\approx$  (*PFn* *x* *T* *B*)  
| *fn2*: [*a*  $\neq$  *b*; *A*  $\approx$  [*a*  $\leftrightarrow$  *b*] · *B*; *a*  $\notin$  *ptrm-fvs* *B*]  $\Longrightarrow$  (*PFn* *a* *T* *A*)  $\approx$  (*PFn* *b* *T* *B*)  
| *pair*: [*A*  $\approx$  *B*; *C*  $\approx$  *D*]  $\Longrightarrow$  (*PPair* *A* *C*)  $\approx$  (*PPair* *B* *D*)  
| *fst*: *A*  $\approx$  *B*  $\Longrightarrow$  *PFst* *A*  $\approx$  *PFst* *B*  
| *snd*: *A*  $\approx$  *B*  $\Longrightarrow$  *PSnd* *A*  $\approx$  *PSnd* *B*

**inductive-cases** *unitE*: *PUnit*  $\approx$  *Y*  
**inductive-cases** *varE*: *PVar* *x*  $\approx$  *Y*  
**inductive-cases** *appE*: *PApp* *A* *B*  $\approx$  *Y*  
**inductive-cases** *fnE*: *PFn* *x* *T* *A*  $\approx$  *Y*  
**inductive-cases** *pairE*: *PPair* *A* *B*  $\approx$  *Y*  
**inductive-cases** *fstE*: *PFst* *P*  $\approx$  *Y*  
**inductive-cases** *sndE*: *PSnd* *P*  $\approx$  *Y*

**lemma** *ptrm-prm-apply-id*:  
**shows**  $\varepsilon \cdot X = X$   
**by**(*induction* *X*, *simp-all* *add*: *prm-apply-id*)

**lemma** *ptrm-prm-apply-compose*:  
**shows**  $\pi \cdot \sigma \cdot X = (\pi \diamond \sigma) \cdot X$   
**by**(*induction* *X*, *simp-all* *add*: *prm-apply-composition*)

```

lemma ptrm-size-prm:
  shows size X = size (π · X)
by(induction X, auto)

lemma ptrm-size-alpha-equiv:
  assumes X ≈ Y
  shows size X = size Y
using assms proof(induction rule: ptrm-alpha-equiv.induct)
  case (fn2 a b A B T)
    hence size A = size B using ptrm-size-prm by metis
    thus ?case by auto
  next
qed auto

lemma ptrm-fvs-finite:
  shows finite (ptrm-fvs X)
by(induction X, auto)

lemma ptrm-prm-fvs:
  shows ptrm-fvs (π · X) = π {$_} ptrm-fvs X
proof(induction X)
  case (PUnit)
    thus ?case unfolding prm-set-def by simp
  next
  case (PVar x)
    have ptrm-fvs (π · PVar x) = ptrm-fvs (PVar (π $ x)) by simp
    moreover have ... = {π $ x} by simp
    moreover have ... = π {$_} {x} using prm-set-singleton by metis
    moreover have ... = π {$_} ptrm-fvs (PVar x) by simp
    ultimately show ?case by metis
  next
  case (PApp A B)
    have ptrm-fvs (π · PApp A B) = ptrm-fvs (PApp (π · A) (π · B)) by simp
    moreover have ... = ptrm-fvs (π · A) ∪ ptrm-fvs (π · B) by simp
    moreover have ... = π {$_} ptrm-fvs A ∪ π {$_} ptrm-fvs B using PApp.IH
by metis
    moreover have ... = π {$_} (ptrm-fvs A ∪ ptrm-fvs B) using prm-set-distributes-union
by metis
    moreover have ... = π {$_} ptrm-fvs (PApp A B) by simp
    ultimately show ?case by metis
  next
  case (PFn x T A)
    have ptrm-fvs (π · PFn x T A) = ptrm-fvs (PFn (π $ x) T (π · A)) by simp
    moreover have ... = ptrm-fvs (π · A) - {π $ x} by simp
    moreover have ... = π {$_} ptrm-fvs A - {π $ x} using PFn.IH by metis
    moreover have ... = π {$_} ptrm-fvs A - π {$_} {x} using prm-set-singleton
by metis
    moreover have ... = π {$_} (ptrm-fvs A - {x}) using prm-set-distributes-difference

```

```

by metis
  moreover have ... =  $\pi$   $\{\$\}$  ptrm-fvs (PFn x T A) by simp
  ultimately show ?case by metis
next
case (PPair A B)
  thus ?case
    using prm-set-distributes-union ptrm-apply-prm.simps(5) ptrm-fvs.simps(5)
    by fastforce
next
case (PFst P)
  thus ?case by auto
next
case (PSnd P)
  thus ?case by auto
next
qed

lemma ptrm-alpha-equiv-fvs:
  assumes  $X \approx Y$ 
  shows ptrm-fvs X = ptrm-fvs Y
using assms proof(induction rule: ptrm-alpha-equiv.induct)
  case (fn2 a b A B T)
    have ptrm-fvs (PFn a T A) = ptrm-fvs A - {a} by simp
    moreover have ... = ptrm-fvs ([a  $\leftrightarrow$  b]  $\cdot$  B) - {a} using fn2.IH by metis
    moreover have ... = ([a  $\leftrightarrow$  b]  $\{\$\}$  ptrm-fvs B) - {a} using ptrm-prm-fvs by
metis
    moreover have ... = ptrm-fvs B - {b} proof -
      consider  $b \in \text{ptrm-fvs } B \mid b \notin \text{ptrm-fvs } B$  by auto
      thus ?thesis proof(cases)
        case 1
          have [a  $\leftrightarrow$  b]  $\{\$\}$  ptrm-fvs B - {a} = [b  $\leftrightarrow$  a]  $\{\$\}$  ptrm-fvs B - {a}
using prm-unit-commutes by metis
          moreover have ... = ((ptrm-fvs B - {b})  $\cup$  {a}) - {a}
            using prm-set-unit-action  $\langle b \in \text{ptrm-fvs } B \rangle \langle a \notin \text{ptrm-fvs } B \rangle$  by metis
          moreover have ... = ptrm-fvs B - {b} using  $\langle a \notin \text{ptrm-fvs } B \rangle \langle a \neq b \rangle$ 
            using Diff-insert0 Diff-insert2 Un-insert-right insert-Diff1 insert-is-Un
singletonI
            sup-bot.right-neutral by blast
          ultimately show ?thesis by metis
        next
        case 2
          hence [a  $\leftrightarrow$  b]  $\{\$\}$  ptrm-fvs B - {a} = ptrm-fvs B - {a}
            using prm-set-unit-inaction  $\langle a \notin \text{ptrm-fvs } B \rangle$  by metis
          moreover have ... = ptrm-fvs B using  $\langle a \notin \text{ptrm-fvs } B \rangle$  by simp
          moreover have ... = ptrm-fvs B - {b} using  $\langle b \notin \text{ptrm-fvs } B \rangle$  by simp
          ultimately show ?thesis by metis
      next
      qed
    next
    qed
  qed

```

```

    moreover have ... = ptrm-fvs (PFn b T B) by simp
    ultimately show ?case by metis
  next
qed auto

lemma ptrm-alpha-equiv-prm:
  assumes  $X \approx Y$ 
  shows  $\pi \cdot X \approx \pi \cdot Y$ 
using assms proof (induction rule: ptrm-alpha-equiv.induct)
  case (unit)
    thus ?case using ptrm-alpha-equiv.unit by simp
  next
  case (var x)
    thus ?case using ptrm-alpha-equiv.var by simp
  next
  case (app A B C D)
    thus ?case using ptrm-alpha-equiv.app by simp
  next
  case (fn1 A B x T)
    thus ?case using ptrm-alpha-equiv.fn1 by simp
  next
  case (fn2 a b A B T)
    have  $\pi \$ a \neq \pi \$ b$  using  $\langle a \neq b \rangle$  using prm-apply-unequal by metis
    moreover have  $\pi \$ a \notin \text{ptrm-fvs } (\pi \cdot B)$  using  $\langle a \notin \text{ptrm-fvs } B \rangle$ 
    using imageE prm-apply-unequal prm-set-def ptrm-prm-fvs by (metis (no-types,
lifting))
    moreover have  $\pi \cdot A \approx [\pi \$ a \leftrightarrow \pi \$ b] \cdot \pi \cdot B$ 
      using fn2.IH prm-compose-push ptrm-prm-apply-compose by metis
    ultimately show ?case using ptrm-alpha-equiv.fn2 by simp
  next
  case (pair A B C D)
    thus ?case using ptrm-alpha-equiv.pair by simp
  next
  case (fst A B)
    thus ?case using ptrm-alpha-equiv.fst by simp
  next
  case (snd A B)
    thus ?case using ptrm-alpha-equiv.snd by simp
  next
qed

lemma ptrm-swp-transfer:
  shows  $[a \leftrightarrow b] \cdot X \approx Y \iff X \approx [a \leftrightarrow b] \cdot Y$ 
proof -
  have 1:  $[a \leftrightarrow b] \cdot X \approx Y \implies X \approx [a \leftrightarrow b] \cdot Y$ 
proof -
  assume  $[a \leftrightarrow b] \cdot X \approx Y$ 
  hence  $\varepsilon \cdot X \approx [a \leftrightarrow b] \cdot Y$ 
  using ptrm-alpha-equiv-prm ptrm-prm-apply-compose prm-unit-involution by

```



*metis*  
**thus** *?thesis using ptrm-prm-apply-id by metis*  
**qed**  
**have**  $2: X \approx [a \leftrightarrow b] \cdot Y \implies [a \leftrightarrow b] \cdot X \approx Y$   
**proof** –  
**assume**  $X \approx [a \leftrightarrow b] \cdot Y$   
**hence**  $[a \leftrightarrow b] \cdot X \approx \varepsilon \cdot Y$   
**using** *ptrm-alpha-equiv-prm ptrm-prm-apply-compose prm-unit-involution by*  
*metis*  
**thus** *?thesis using ptrm-prm-apply-id by metis*  
**qed**  
**from** 1 and 2 **show**  $[a \leftrightarrow b] \cdot X \approx Y \longleftrightarrow X \approx [a \leftrightarrow b] \cdot Y$  **by** *blast*  
**qed**

**lemma** *ptrm-alpha-equiv-fvs-transfer:*  
**assumes**  $A \approx [a \leftrightarrow b] \cdot B$  **and**  $a \notin \text{ptrm-fvs } B$   
**shows**  $b \notin \text{ptrm-fvs } A$   
**proof** –  
**from**  $\langle A \approx [a \leftrightarrow b] \cdot B \rangle$  **have**  $[a \leftrightarrow b] \cdot A \approx B$  **using** *ptrm-swp-transfer by*  
*metis*  
**hence**  $\text{ptrm-fvs } B = [a \leftrightarrow b] \{\$\} \text{ptrm-fvs } A$   
**using** *ptrm-alpha-equiv-fvs ptrm-prm-fvs by metis*  
**hence**  $a \notin [a \leftrightarrow b] \{\$\} \text{ptrm-fvs } A$  **using**  $\langle a \notin \text{ptrm-fvs } B \rangle$  **by** *metis*  
**hence**  $b \notin [a \leftrightarrow b] \{\$\} ([a \leftrightarrow b] \{\$\} \text{ptrm-fvs } A)$   
**using** *prn-set-notmembership prm-unit-action by metis*  
**thus** *?thesis using prn-set-apply-compose prm-unit-involution prm-set-id by*  
*metis*  
**qed**

**lemma** *ptrm-prm-agreement-equiv:*  
**assumes**  $\bigwedge a. a \in \text{ds } \pi \sigma \implies a \notin \text{ptrm-fvs } M$   
**shows**  $\pi \cdot M \approx \sigma \cdot M$   
**using** *assms proof(induction M arbitrary:  $\pi \sigma$ )*  
**case** (*PUnit*)  
**thus** *?case using ptrm-alpha-equiv.unit by simp*  
**next**  
**case** (*PVar x*)  
**consider**  $x \in \text{ds } \pi \sigma \mid x \notin \text{ds } \pi \sigma$  **by** *auto*  
**thus** *?case proof(cases)*  
**case** 1  
**hence**  $x \notin \text{ptrm-fvs } (\text{PVar } x)$  **using** *PVar.premis by blast*  
**thus** *?thesis by simp*  
**next**  
**case** 2  
**hence**  $\pi \$ x = \sigma \$ x$  **using** *prn-disagreement-ext by metis*  
**thus** *?thesis using ptrm-alpha-equiv.var by simp*  
**next**  
**qed**  
**next**

**case** ( $PApp\ A\ B$ )  
**hence**  $\pi \cdot A \approx \sigma \cdot A$  **and**  $\pi \cdot B \approx \sigma \cdot B$  **by** *auto*  
**thus** *?case* **using** *ptrm-alpha-equiv.app* **by** *auto*  
**next**  
**case** ( $PFn\ x\ T\ A$ )  
**consider**  $x \notin ds\ \pi\ \sigma \mid x \in ds\ \pi\ \sigma$  **by** *auto*  
**thus** *?case* **proof**(*cases*)  
**case** 1  
**hence**  $\ast: \pi\ \$\ x = \sigma\ \$\ x$  **using** *prm-disagreement-ext* **by** *metis*  
**have**  $\bigwedge a. a \in ds\ \pi\ \sigma \implies a \notin ptrm-fvs\ A$   
**proof** –  
**fix**  $a$   
**assume**  $a \in ds\ \pi\ \sigma$   
**hence**  $a \notin ptrm-fvs\ (PFn\ x\ T\ A)$  **using** *PFn.prem*s **by** *metis*  
**hence**  $a = x \vee a \notin ptrm-fvs\ A$  **by** *auto*  
**thus**  $a \notin ptrm-fvs\ A$  **using**  $\langle a \in ds\ \pi\ \sigma \rangle \langle x \notin ds\ \pi\ \sigma \rangle$  **by** *auto*  
**qed**  
**thus** *?thesis* **using** *PFn.IH*  $\ast$  *ptrm-alpha-equiv.fn1* *ptrm-apply-prm.simps*(3)  
**by** *fastforce*  
**next**  
**case** 2  
**hence**  $\pi\ \$\ x \neq \sigma\ \$\ x$  **using** *prm-disagreement-def* *CollectD* **by** *fastforce*  
**moreover** **have**  $\pi\ \$\ x \notin ptrm-fvs\ (\sigma \cdot A)$   
**proof** –  
**have**  $y \in (ptrm-fvs\ A) \implies \pi\ \$\ x \neq \sigma\ \$\ y$  **for**  $y$   
**using** *PFn*  $\langle \pi\ \$\ x \neq \sigma\ \$\ x \rangle$  *prm-apply-unequal* *prm-disagreement-ext*  
*ptrm-fvs.simps*(4)  
**by** (*metis* *Diff-iff* *empty-iff* *insert-iff*)  
**hence**  $\pi\ \$\ x \notin \sigma\ \$\ \{\}$  *ptrm-fvs*  $A$  **unfolding** *prm-set-def* **by** *auto*  
**thus** *?thesis* **using** *ptrm-prm-fvs* **by** *metis*  
**qed**  
**moreover** **have**  $\pi \cdot A \approx [\pi\ \$\ x \leftrightarrow \sigma\ \$\ x] \cdot \sigma \cdot A$   
**proof** –  
**have**  $\bigwedge a. a \in ds\ \pi\ ([\pi\ \$\ x \leftrightarrow \sigma\ \$\ x] \diamond \sigma) \implies a \notin ptrm-fvs\ A$  **proof** –  
**fix**  $a$   
**assume**  $\ast: a \in ds\ \pi\ ([\pi\ \$\ x \leftrightarrow \sigma\ \$\ x] \diamond \sigma)$   
**hence**  $a \neq x$  **using**  $\langle \pi\ \$\ x \neq \sigma\ \$\ x \rangle$   
**using** *prm-apply-composition* *prm-disagreement-ext* *prm-unit-action*  
*prm-unit-commutes*  
**by** *metis*  
**hence**  $a \in ds\ \pi\ \sigma$   
**using**  $\ast$  *prm-apply-composition* *prm-apply-unequal* *prm-disagreement-ext*  
*prm-unit-inaction*  
**by** *metis*  
**thus**  $a \notin ptrm-fvs\ A$  **using**  $\langle a \neq x \rangle$  *PFn.prem*s **by** *auto*  
**qed**  
**thus** *?thesis* **using** *PFn* **by** (*simp* *add: prm-prm-apply-compose*)  
**qed**  
**ultimately** **show** *?thesis* **using** *ptrm-alpha-equiv.fn2* **by** *simp*

```

    next
  qed
next
case (PPair A B)
  hence  $\pi \cdot A \approx \sigma \cdot A$  and  $\pi \cdot B \approx \sigma \cdot B$  by auto
  thus ?case using ptrm-alpha-equiv.pair by auto
next
case (PFst P)
  hence  $\pi \cdot P \approx \sigma \cdot P$  by auto
  thus ?case using ptrm-alpha-equiv.fst by auto
next
case (PSnd P)
  hence  $\pi \cdot P \approx \sigma \cdot P$  by auto
  thus ?case using ptrm-alpha-equiv.snd by auto
next
qed

```

**lemma** *ptrm-prm-unit-inaction*:

```

  assumes  $a \notin \text{ptrm-fvs } X$   $b \notin \text{ptrm-fvs } X$ 
  shows  $[a \leftrightarrow b] \cdot X \approx X$ 
proof –
  have  $(\bigwedge x. x \in \text{ds } [a \leftrightarrow b] \varepsilon \implies x \notin \text{ptrm-fvs } X)$ 
  proof –
    fix  $x$ 
    assume  $x \in \text{ds } [a \leftrightarrow b] \varepsilon$ 
    hence  $[a \leftrightarrow b] \$ x \neq \varepsilon \$ x$ 
      unfolding prm-disagreement-def
      by auto
    hence  $x = a \vee x = b$ 
      using prm-apply-id prm-unit-inaction by metis
    thus  $x \notin \text{ptrm-fvs } X$  using assms by auto
  qed
  hence  $[a \leftrightarrow b] \cdot X \approx \varepsilon \cdot X$ 
    using ptrm-prm-agreement-equiv by metis
  thus ?thesis using ptrm-prm-apply-id by metis
qed

```

**lemma** *ptrm-alpha-equiv-reflexive*:

```

  shows  $M \approx M$ 
by(induction M, auto simp add: ptrm-alpha-equiv.intros)

```

**corollary** *ptrm-alpha-equiv-reflp*:

```

  shows reflp ptrm-alpha-equiv
unfolding reflp-def using ptrm-alpha-equiv-reflexive by auto

```

**lemma** *ptrm-alpha-equiv-symmetric*:

```

  assumes  $X \approx Y$ 
  shows  $Y \approx X$ 
using assms proof(induction rule: ptrm-alpha-equiv.induct, auto simp add: ptrm-alpha-equiv.intros)

```

```

case (fn2 a b A B T)
  have  $b \neq a$  using  $\langle a \neq b \rangle$  by auto
  have  $B \approx [b \leftrightarrow a] \cdot A$  using  $\langle [a \leftrightarrow b] \cdot B \approx A \rangle$ 
    using ptrm-sup-transfer ptrm-unit-commutes by metis

  have  $b \notin \text{ptrm-fvs } A$  using  $\langle a \notin \text{ptrm-fvs } B \rangle \langle A \approx [a \leftrightarrow b] \cdot B \rangle \langle a \neq b \rangle$ 
    using ptrm-alpha-equiv-fvs-transfer by metis

  show ?case using  $\langle b \neq a \rangle \langle B \approx [b \leftrightarrow a] \cdot A \rangle \langle b \notin \text{ptrm-fvs } A \rangle$ 
    using ptrm-alpha-equiv.fn2 by metis
next
qed

corollary ptrm-alpha-equiv-symp:
  shows symp ptrm-alpha-equiv
unfolding symp-def using ptrm-alpha-equiv-symmetric by auto

lemma ptrm-alpha-equiv-transitive:
  assumes  $X \approx Y$  and  $Y \approx Z$ 
  shows  $X \approx Z$ 
using assms proof(induction size X arbitrary: X Y Z rule: less-induct)
  fix X Y Z :: 'a ptrm
  assume IH:  $\bigwedge K Y Z :: 'a \text{ ptrm. size } K < \text{size } X \implies K \approx Y \implies Y \approx Z \implies K \approx Z$ 
  assume  $X \approx Y$  and  $Y \approx Z$ 
  show  $X \approx Z$  proof(cases X)
    case (PUnit)
      hence  $Y = PUnit$  using  $\langle X \approx Y \rangle$  unitE by metis
      hence  $Z = PUnit$  using  $\langle Y \approx Z \rangle$  unitE by metis
      thus ?thesis using ptrm-alpha-equiv.unit  $\langle X = PUnit \rangle$  by metis
    next
    case (PVar x)
      hence  $PVar x \approx Y$  using  $\langle X \approx Y \rangle$  by auto
      hence  $Y = PVar x$  using varE by metis
      hence  $PVar x \approx Z$  using  $\langle Y \approx Z \rangle$  by auto
      hence  $Z = PVar x$  using varE by metis
      thus ?thesis using ptrm-alpha-equiv.var  $\langle X = PVar x \rangle$  by metis
    next
    case (PApp A B)
      obtain C D where  $Y = PApp C D$  and  $A \approx C$  and  $B \approx D$ 
        using appE  $\langle X = PApp A B \rangle \langle X \approx Y \rangle$  by metis
      hence  $PApp C D \approx Z$  using  $\langle Y \approx Z \rangle$  by simp
      from this obtain E F where  $Z = PApp E F$  and  $C \approx E$  and  $D \approx F$  using
        appE by metis

      have  $\text{size } A < \text{size } X$  and  $\text{size } B < \text{size } X$  using  $\langle X = PApp A B \rangle$  by auto
      hence  $A \approx E$  and  $B \approx F$  using IH  $\langle A \approx C \rangle \langle C \approx E \rangle \langle B \approx D \rangle \langle D \approx F \rangle$  by
        auto
      thus ?thesis using  $\langle X = PApp A B \rangle \langle Z = PApp E F \rangle$  ptrm-alpha-equiv.app

```

**by** *metis*  
**next**  
**case**  $\langle PFn\ x\ T\ A \rangle$   
**from** *this* **have**  $X: X = PFn\ x\ T\ A.$   
**hence**  $*$ :  $size\ A < size\ X$  **by** *auto*  
  
**obtain**  $y\ B$  **where**  $Y = PFn\ y\ T\ B$   
**and**  $Y$ -cases:  $(x = y \wedge A \approx B) \vee (x \neq y \wedge A \approx [x \leftrightarrow y] \cdot B \wedge x \notin ptrm\text{-}fvs\ B)$   
**using**  $fnE\ \langle X \approx Y \rangle\ \langle X = PFn\ x\ T\ A \rangle$  **by** *metis*  
**obtain**  $z\ C$  **where**  $Z: Z = PFn\ z\ T\ C$   
**and**  $Z$ -cases:  $(y = z \wedge B \approx C) \vee (y \neq z \wedge B \approx [y \leftrightarrow z] \cdot C \wedge y \notin ptrm\text{-}fvs\ C)$   
**using**  $fnE\ \langle Y \approx Z \rangle\ \langle Y = PFn\ y\ T\ B \rangle$  **by** *metis*  
  
**consider**  
 $x = y\ A \approx B$  **and**  $y = z\ B \approx C$   
 $| x = y\ A \approx B$  **and**  $y \neq z\ B \approx [y \leftrightarrow z] \cdot C$   $y \notin ptrm\text{-}fvs\ C$   
 $| x \neq y\ A \approx [x \leftrightarrow y] \cdot B$   $x \notin ptrm\text{-}fvs\ B$  **and**  $y = z\ B \approx C$   
 $| x \neq y\ A \approx [x \leftrightarrow y] \cdot B$   $x \notin ptrm\text{-}fvs\ B$  **and**  $y \neq z\ B \approx [y \leftrightarrow z] \cdot C$   $y \notin ptrm\text{-}fvs\ C$  **and**  $x = z$   
 $| x \neq y\ A \approx [x \leftrightarrow y] \cdot B$   $x \notin ptrm\text{-}fvs\ B$  **and**  $y \neq z\ B \approx [y \leftrightarrow z] \cdot C$   $y \notin ptrm\text{-}fvs\ C$  **and**  $x \neq z$   
**using**  $Y$ -cases  $Z$ -cases **by** *auto*  
  
**thus** *?thesis* **proof**(cases)  
**case** 1  
**have**  $A \approx C$  **using**  $\langle A \approx B \rangle\ \langle B \approx C \rangle$   $IH\ *$  **by** *metis*  
**have**  $x = z$  **using**  $\langle x = y \rangle\ \langle y = z \rangle$  **by** *metis*  
**show** *?thesis* **using**  $\langle A \approx C \rangle\ \langle x = z \rangle$   $X\ Z$   
**using** *ptrm-alpha-equiv.fn1* **by** *metis*  
**next**  
**case** 2  
**have**  $x \neq z$  **using**  $\langle x = y \rangle\ \langle y \neq z \rangle$  **by** *metis*  
**have**  $A \approx [x \leftrightarrow z] \cdot C$  **using**  $\langle A \approx B \rangle\ \langle B \approx [y \leftrightarrow z] \cdot C \rangle\ \langle x = y \rangle$   $IH\ *$   
**by** *metis*  
**have**  $x \notin ptrm\text{-}fvs\ C$  **using**  $\langle x = y \rangle\ \langle y \notin ptrm\text{-}fvs\ C \rangle$  **by** *metis*  
**thus** *?thesis* **using**  $\langle x \neq z \rangle\ \langle A \approx [x \leftrightarrow z] \cdot C \rangle\ \langle x \notin ptrm\text{-}fvs\ C \rangle$   $X\ Z$   
**using** *ptrm-alpha-equiv.fn2* **by** *metis*  
**next**  
**case** 3  
**have**  $x \neq z$  **using**  $\langle x \neq y \rangle\ \langle y = z \rangle$  **by** *metis*  
**have**  $[x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot C$  **using**  $\langle B \approx C \rangle$  *ptrm-alpha-equiv-prm*  
**by** *metis*  
**have**  $A \approx [x \leftrightarrow z] \cdot C$   
**using**  $\langle A \approx [x \leftrightarrow y] \cdot B \rangle\ \langle [x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot C \rangle\ \langle y = z \rangle$   $IH\ *$   
**by** *metis*  
**have**  $x \notin ptrm\text{-}fvs\ C$  **using**  $\langle B \approx C \rangle\ \langle x \notin ptrm\text{-}fvs\ B \rangle$  *ptrm-alpha-equiv-fvs*  
**by** *metis*

**thus** *?thesis* **using**  $\langle x \neq z \rangle \langle A \approx [x \leftrightarrow z] \cdot C \rangle \langle x \notin \text{ptrm-fvs } C \rangle X Z$   
**using** *ptrm-alpha-equiv.fn2* **by** *metis*  
**next**  
**case 4**  
**have**  $[x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C$   
**using**  $\langle B \approx [y \leftrightarrow z] \cdot C \rangle$  *ptrm-alpha-equiv-prm* **by** *metis*  
**hence**  $A \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C$   
**using**  $\langle A \approx [x \leftrightarrow y] \cdot B \rangle$  *IH* \* **by** *metis*  
**hence**  $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C$  **using** *ptrm-prm-apply-compose* **by**  
*metis*  
**hence**  $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow x]) \cdot C$  **using**  $\langle x = z \rangle$  **by** *metis*  
**hence**  $A \approx ([x \leftrightarrow y] \diamond [x \leftrightarrow y]) \cdot C$  **using** *prn-unit-commutes* **by** *metis*  
**hence**  $A \approx \varepsilon \cdot C$  **using**  $\langle x = z \rangle$  *prn-unit-involution* **by** *metis*  
**hence**  $A \approx C$  **using** *ptrm-prm-apply-id* **by** *metis*  
  
**thus** *?thesis* **using**  $\langle x = z \rangle \langle A \approx C \rangle X Z$   
**using** *ptrm-alpha-equiv.fn1* **by** *metis*  
**next**  
**case 5**  
**have**  $x \notin \text{ptrm-fvs } C$  **proof** –  
**have**  $\text{ptrm-fvs } B = \text{ptrm-fvs } ([y \leftrightarrow z] \cdot C)$   
**using** *ptrm-alpha-equiv-fvs*  $\langle B \approx [y \leftrightarrow z] \cdot C \rangle$  **by** *metis*  
**hence**  $x \notin \text{ptrm-fvs } ([y \leftrightarrow z] \cdot C)$  **using**  $\langle x \notin \text{ptrm-fvs } B \rangle$  **by** *auto*  
**hence**  $x \notin [y \leftrightarrow z] \{ \$ \} \text{ptrm-fvs } C$  **using** *ptrm-prm-fvs* **by** *metis*  
**consider**  $z \in \text{ptrm-fvs } C \mid z \notin \text{ptrm-fvs } C$  **by** *auto*  
**thus** *?thesis* **proof**(*cases*)  
**case 1**  
**hence**  $[y \leftrightarrow z] \{ \$ \} \text{ptrm-fvs } C = \text{ptrm-fvs } C - \{z\} \cup \{y\}$   
**using** *prn-set-unit-action* *prn-unit-commutes*  
**and**  $\langle z \in \text{ptrm-fvs } C \rangle \langle y \notin \text{ptrm-fvs } C \rangle$  **by** *metis*  
**hence**  $x \notin \text{ptrm-fvs } C - \{z\} \cup \{y\}$  **using**  $\langle x \notin [y \leftrightarrow z] \{ \$ \} \text{ptrm-fvs } C \rangle$  **by** *auto*  
**hence**  $x \notin \text{ptrm-fvs } C - \{z\}$  **using**  $\langle x \neq y \rangle$  **by** *auto*  
**thus** *?thesis* **using**  $\langle x \neq z \rangle$  **by** *auto*  
**next**  
**case 2**  
**hence**  $[y \leftrightarrow z] \{ \$ \} \text{ptrm-fvs } C = \text{ptrm-fvs } C$  **using** *prn-set-unit-inaction*  
 $\langle y \notin \text{ptrm-fvs } C \rangle$  **by** *metis*  
**thus** *?thesis* **using**  $\langle x \notin [y \leftrightarrow z] \{ \$ \} \text{ptrm-fvs } C \rangle$  **by** *auto*  
**next**  
**qed**  
**qed**  
  
**have**  $A \approx [x \leftrightarrow z] \cdot C$  **proof** –  
**have**  $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C$   
**using** *IH* \*  $\langle A \approx [x \leftrightarrow y] \cdot B \rangle \langle B \approx [y \leftrightarrow z] \cdot C \rangle$   
**and** *ptrm-alpha-equiv-prm* *ptrm-prm-apply-compose* **by** *metis*  
  
**have**  $([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C \approx [x \leftrightarrow z] \cdot C$  **proof** –

**have**  $ds ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) [x \leftrightarrow z] = \{x, y\}$   
**using**  $\langle x \neq y \rangle \langle y \neq z \rangle \langle x \neq z \rangle$  *prm-disagreement-composition* **by** *metis*

**hence**  $\bigwedge a. a \in ds ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) [x \leftrightarrow z] \implies a \notin ptrm-fvs C$   
**using**  $\langle x \notin ptrm-fvs C \rangle \langle y \notin ptrm-fvs C \rangle$   
**using** *Diff-iff Diff-insert-absorb insert-iff* **by** *auto*  
**thus** *?thesis* **using** *ptrm-prm-agreement-equiv* **by** *metis*  
**qed**

**thus** *?thesis* **using** *IH* \*  
**using**  $\langle A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C \rangle \langle ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C \approx [x \leftrightarrow z] \cdot C \rangle$   
**by** *metis*  
**qed**

**show** *?thesis* **using**  $\langle x \neq z \rangle \langle A \approx [x \leftrightarrow z] \cdot C \rangle \langle x \notin ptrm-fvs C \rangle X Z$   
**using** *ptrm-alpha-equiv.fn2* **by** *metis*

**next**  
**qed**  
**next**  
**case** (*PPair A B*)  
**obtain**  $C D$  **where**  $Y = PPair C D$  **and**  $A \approx C$  **and**  $B \approx D$   
**using** *pairE*  $\langle X = PPair A B \rangle \langle X \approx Y \rangle$  **by** *metis*  
**hence**  $PPair C D \approx Z$  **using**  $\langle Y \approx Z \rangle$  **by** *simp*  
**from this** **obtain**  $E F$  **where**  $Z = PPair E F$  **and**  $C \approx E$  **and**  $D \approx F$   
**using** *pairE* **by** *metis*

**have**  $size A < size X$  **and**  $size B < size X$  **using**  $\langle X = PPair A B \rangle$  **by** *auto*  
**hence**  $A \approx E$  **and**  $B \approx F$  **using** *IH*  $\langle A \approx C \rangle \langle C \approx E \rangle \langle B \approx D \rangle \langle D \approx F \rangle$  **by** *auto*  
**thus** *?thesis* **using**  $\langle X = PPair A B \rangle \langle Z = PPair E F \rangle$  *ptrm-alpha-equiv.pair* **by** *metis*

**next**  
**case** (*PFst P*)  
**obtain**  $Q$  **where**  $Y = PFst Q P \approx Q$  **using** *fstE*  $\langle X = PFst P \rangle \langle X \approx Y \rangle$   
**by** *metis*  
**obtain**  $R$  **where**  $Z = PFst R Q \approx R$  **using** *fstE*  $\langle Y = PFst Q \rangle \langle Y \approx Z \rangle$  **by** *metis*

**have**  $size P < size X$  **using**  $\langle X = PFst P \rangle$  **by** *auto*  
**hence**  $P \approx R$  **using** *IH*  $\langle P \approx Q \rangle \langle Q \approx R \rangle$  **by** *metis*  
**thus** *?thesis* **using**  $\langle X = PFst P \rangle \langle Z = PFst R \rangle$  *ptrm-alpha-equiv.fst* **by** *metis*

**next**  
**case** (*PSnd P*)  
**obtain**  $Q$  **where**  $Y = PSnd Q P \approx Q$  **using** *sndE*  $\langle X = PSnd P \rangle \langle X \approx Y \rangle$   
**by** *metis*  
**obtain**  $R$  **where**  $Z = PSnd R Q \approx R$  **using** *sndE*  $\langle Y = PSnd Q \rangle \langle Y \approx Z \rangle$   
**by** *metis*

```

    have size P < size X using ⟨X = PSnd P⟩ by auto
    hence P ≈ R using IH ⟨P ≈ Q⟩ ⟨Q ≈ R⟩ by metis
    thus ?thesis using ⟨X = PSnd P⟩ ⟨Z = PSnd R⟩ ptrm-alpha-equiv.snd by
metis
  next
  qed
qed

```

```

corollary ptrm-alpha-equiv-transp:
  shows transp ptrm-alpha-equiv
unfolding transp-def using ptrm-alpha-equiv-transitive by auto

```

```

type-synonym 'a typing-ctx = 'a → type

```

```

fun ptrm-infer-type :: 'a typing-ctx ⇒ 'a ptrm ⇒ type option where
  ptrm-infer-type Γ PUnit = Some TUnit
| ptrm-infer-type Γ (PVar x) = Γ x
| ptrm-infer-type Γ (PApp A B) = (case (ptrm-infer-type Γ A, ptrm-infer-type Γ
B) of
  (Some (TArr τ σ), Some τ') ⇒ (if τ = τ' then Some σ else None)
  | - ⇒ None
  )
| ptrm-infer-type Γ (PFn x τ A) = (case ptrm-infer-type (Γ(x ↦ τ)) A of
  Some σ ⇒ Some (TArr τ σ)
  | None ⇒ None
  )
| ptrm-infer-type Γ (PPair A B) = (case (ptrm-infer-type Γ A, ptrm-infer-type Γ
B) of
  (Some τ, Some σ) ⇒ Some (TPair τ σ)
  | - ⇒ None
  )
| ptrm-infer-type Γ (PFst P) = (case ptrm-infer-type Γ P of
  (Some (TPair τ σ)) ⇒ Some τ
  | - ⇒ None
  )
| ptrm-infer-type Γ (PSnd P) = (case ptrm-infer-type Γ P of
  (Some (TPair τ σ)) ⇒ Some σ
  | - ⇒ None
  )

```

```

lemma ptrm-infer-type-swp-types:
  assumes a ≠ b
  shows ptrm-infer-type (Γ(a ↦ T, b ↦ S)) X = ptrm-infer-type (Γ(a ↦ S, b ↦
T)) ([a ↔ b] · X)
using assms proof(induction X arbitrary: T S Γ)
  case (PUnit)
  thus ?case by simp
  next

```



```

case (PVar x)
  consider x = a | x = b | x ≠ a ∧ x ≠ b by auto
  thus ?case proof(cases)
    assume x = a
    thus ?thesis using ⟨a ≠ b⟩ by (simp add: prn-unit-action)
    next

    assume x = b
    thus ?thesis using ⟨a ≠ b⟩
      using prn-unit-action prn-unit-commutes fun-upd-same fun-upd-twist
      by (metis prn-apply-prn.simps(2) prn-infer-type.simps(2))
    next

    assume x ≠ a ∧ x ≠ b
    thus ?thesis by (simp add: prn-unit-inaction)
    next
  qed
next
case (PApp A B)
  thus ?case by simp
next
case (PFn x τ A)
  hence *:
    prn-infer-type (Γ(a ↦ T, b ↦ S)) A = prn-infer-type (Γ(a ↦ S, b ↦ T))
    ([a ↔ b] · A)
  for T S Γ
  by metis

  consider x = a | x = b | x ≠ a ∧ x ≠ b by auto
  thus ?case proof(cases)
    case 1
      hence
        prn-infer-type (Γ(a ↦ S, b ↦ T)) ([a ↔ b] · PFn x τ A)
        = prn-infer-type (Γ(a ↦ S, b ↦ T)) (PFn b τ ([a ↔ b] · A))
          using prn-unit-action prn-apply-prn.simps(4) by metis
        moreover have ... = (case prn-infer-type (Γ(a ↦ S, b ↦ τ)) ([a ↔ b] ·
        A) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
          by simp
        moreover have ... = (case prn-infer-type (Γ(a ↦ τ, b ↦ S)) A of None
        ⇒ None | Some σ ⇒ Some (TArr τ σ))
          using * by metis
        moreover have ... = (case prn-infer-type (Γ(b ↦ S, a ↦ T, a ↦ τ)) A
        of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
          using ⟨a ≠ b⟩ fun-upd-twist fun-upd-upd by metis
        moreover have ... = prn-infer-type (Γ(b ↦ S, a ↦ T)) (PFn x τ A)
          using ⟨x = a⟩ by simp
        moreover have ... = prn-infer-type (Γ(a ↦ T, b ↦ S)) (PFn x τ A)
          using ⟨a ≠ b⟩ fun-upd-twist by metis
        ultimately show ?thesis by metis

```

**next**  
**case 2**  
**hence**  

$$\begin{aligned} & \text{ptrm-infer-type } (\Gamma(a \mapsto S, b \mapsto T)) ([a \leftrightarrow b] \cdot \text{PFn } x \tau A) \\ & = \text{ptrm-infer-type } (\Gamma(a \mapsto S, b \mapsto T)) (\text{PFn } a \tau ([a \leftrightarrow b] \cdot A)) \\ & \text{using } \text{prm-unit-action } \text{prm-unit-commutes } \text{ptrm-apply-prm.simps}(4) \text{ by} \end{aligned}$$
*metis*  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto S, b \mapsto T)(a \mapsto \tau))$   $([a \leftrightarrow b] \cdot A)$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**by simp**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto \tau, b \mapsto T))$   $([a \leftrightarrow b] \cdot A)$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using fun-upd-upd fun-upd-twist  $\langle a \neq b \rangle$  by metis**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto T, b \mapsto \tau))$   $A$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using \* by metis**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto T, b \mapsto S)(b \mapsto \tau))$   $A$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using  $\langle a \neq b \rangle$  fun-upd-upd by metis**  
**moreover have** ... = ptrm-infer-type  $(\Gamma(b \mapsto S, a \mapsto T))$  (PFn  $x \tau A$ )  
**using  $\langle x = b \rangle$  by simp**  
**moreover have** ... = ptrm-infer-type  $(\Gamma(a \mapsto T, b \mapsto S))$  (PFn  $x \tau A$ )  
**using  $\langle a \neq b \rangle$  fun-upd-twist by metis**  
**ultimately show ?thesis by metis**  
**next**  
**case 3**  
**hence  $x \neq a$   $x \neq b$  by auto**  
**hence**  

$$\begin{aligned} & \text{ptrm-infer-type } (\Gamma(a \mapsto S, b \mapsto T)) ([a \leftrightarrow b] \cdot \text{PFn } x \tau A) \\ & = \text{ptrm-infer-type } (\Gamma(a \mapsto S, b \mapsto T)) (\text{PFn } x \tau ([a \leftrightarrow b] \cdot A)) \\ & \text{by (simp add: prm-unit-inaction)} \end{aligned}$$
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto S, b \mapsto T)(x \mapsto \tau))$   $([a \leftrightarrow b] \cdot A)$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**by simp**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(x \mapsto \tau, a \mapsto S, b \mapsto T))$   $([a \leftrightarrow b] \cdot A)$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using  $\langle x \neq a \rangle \langle x \neq b \rangle$  fun-upd-twist by metis**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(x \mapsto \tau, a \mapsto T, b \mapsto S))$   $A$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using \* by metis**  
**moreover have** ... = (case ptrm-infer-type  $(\Gamma(a \mapsto T, b \mapsto S, x \mapsto \tau))$   $A$  of None  $\Rightarrow$  None | Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ ))  
**using  $\langle x \neq a \rangle \langle x \neq b \rangle$  fun-upd-twist by metis**  
**moreover have** ... = ptrm-infer-type  $(\Gamma(a \mapsto T, b \mapsto S))$  (PFn  $x \tau A$ ) **by**  
*simp*  
**ultimately show ?thesis by metis**  
**next**  
**qed**  
**next**

```

case (PPair A B)
  thus ?case by simp
next
case (PFst P)
  thus ?case by simp
next
case (PSnd P)
  thus ?case by simp
next
qed

lemma ptrm-infer-type-swp:
  assumes  $a \neq b$   $b \notin \text{ptrm-fvs } X$ 
  shows  $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) X = \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot X)$ 
using assms proof(induction X arbitrary:  $\tau \Gamma$ )
  case (PUnit)
    thus ?case by simp
  next
  case (PVar x)
    hence  $x \neq b$  by simp
    consider  $x = a \mid x \neq a$  by auto
    thus ?case proof(cases)
      case 1
        hence  $[a \leftrightarrow b] \cdot (\text{PVar } x) = \text{PVar } b$ 
        and  $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) (\text{PVar } x) = \text{Some } \tau$  using prm-unit-action
      by auto
        thus ?thesis by auto
      next
        case 2
          hence *:  $[a \leftrightarrow b] \cdot \text{PVar } x = \text{PVar } x$  using  $\langle x \neq b \rangle$  prm-unit-inaction by
          simp
          consider  $\exists \sigma. \Gamma x = \text{Some } \sigma \mid \Gamma x = \text{None}$  by auto
          thus ?thesis proof(cases)
            assume  $\exists \sigma. \Gamma x = \text{Some } \sigma$ 
            from this obtain  $\sigma$  where  $\Gamma x = \text{Some } \sigma$  by auto
            thus ?thesis using *  $\langle x \neq a \rangle \langle x \neq b \rangle$  by auto
            next
              assume  $\Gamma x = \text{None}$ 
              thus ?thesis using *  $\langle x \neq a \rangle \langle x \neq b \rangle$  by auto
          qed
        next
          qed
      next
        qed
    next
      case (PApp A B)
        have  $b \notin \text{ptrm-fvs } A$  and  $b \notin \text{ptrm-fvs } B$  using PApp.premis by auto
        hence  $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) A = \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot A)$ 

```

A)  
**and**  $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) B = \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot B)$   
 B)  
**using**  $\text{PApp.IH assms}$  **by**  $\text{metis+}$

**thus**  $?case$  **by**  $(\text{metis ptrm-apply-prm.simps}(3) \text{ ptrm-infer-type.simps}(3))$   
**next**  
**case**  $(\text{PFn } x \sigma A)$   
**consider**  $b \neq x \wedge b \notin \text{ptrm-fvs } A \mid b = x$  **using**  $\text{PFn.premis}$  **by**  $\text{auto}$   
**thus**  $?case$  **proof** $(cases)$   
**case** 1  
**hence**  $b \neq x \wedge b \notin \text{ptrm-fvs } A$  **by**  $\text{auto}$   
**hence**  $*$ :  $\bigwedge \tau \Gamma. \text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) A = \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot A)$   
**using**  $\text{PFn.IH assms}$  **by**  $\text{metis}$   
**consider**  $a = x \mid a \neq x$  **by**  $\text{auto}$   
**thus**  $?thesis$  **proof** $(cases)$   
**case** 1  
**hence**  $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) (\text{PFn } x \sigma A) = \text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) (\text{PFn } a \sigma A)$   
**and**  
 $\text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot \text{PFn } x \sigma A) =$   
 $\text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) (\text{PFn } b \sigma ([a \leftrightarrow b] \cdot A))$   
**by**  $(\text{auto simp add: prm-unit-action})$   
**thus**  $?thesis$  **using**  $*$   $\text{ptrm-infer-type.simps}(4)$   $\text{fun-upd-upd}$  **by**  $\text{metis}$   
**next**

**case** 2  
**hence**  
 $\text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot \text{PFn } x \sigma A)$   
 $= \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) (\text{PFn } x \sigma ([a \leftrightarrow b] \cdot A))$   
**using**  $\langle b \neq x \rangle$  **by**  $(\text{simp add: prm-unit-inaction})$   
**moreover** **have**  $\dots = (\text{case } \text{ptrm-infer-type } (\Gamma(b \mapsto \tau, x \mapsto \sigma)) ([a \leftrightarrow b] \cdot A) \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } \sigma' \Rightarrow \text{Some } (\text{TArr } \sigma \sigma'))$   
**by**  $\text{simp}$   
**moreover** **have**  $\dots = (\text{case } \text{ptrm-infer-type } (\Gamma(x \mapsto \sigma, b \mapsto \tau)) ([a \leftrightarrow b] \cdot A) \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } \sigma' \Rightarrow \text{Some } (\text{TArr } \sigma \sigma'))$   
**using**  $\langle b \neq x \rangle$   $\text{fun-upd-twist}$  **by**  $\text{metis}$   
**moreover** **have**  $\dots = (\text{case } \text{ptrm-infer-type } (\Gamma(x \mapsto \sigma, a \mapsto \tau)) A \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } \sigma' \Rightarrow \text{Some } (\text{TArr } \sigma \sigma'))$   
**using**  $*$  **by**  $\text{metis}$   
**moreover** **have**  $\dots = (\text{case } \text{ptrm-infer-type } (\Gamma(a \mapsto \tau, x \mapsto \sigma)) A \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } \sigma' \Rightarrow \text{Some } (\text{TArr } \sigma \sigma'))$   
**using**  $\langle a \neq x \rangle$   $\text{fun-upd-twist}$  **by**  $\text{metis}$   
**moreover** **have**  $\dots = \text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) (\text{PFn } x \sigma A)$   
**by**  $\text{simp}$   
**ultimately** **show**  $?thesis$  **by**  $\text{metis}$   
**next**  
**qed**

```

next

case 2
  hence  $a \neq x$  using assms by auto
  have
    ptrm-infer-type ( $\Gamma(a \mapsto \tau)(b \mapsto \sigma)$ )  $A =$ 
    ptrm-infer-type ( $\Gamma(b \mapsto \tau)(a \mapsto \sigma)$ ) ( $[a \leftrightarrow b] \cdot A$ )
  using ptrm-infer-type-swp-types using  $\langle a \neq b \rangle$  fun-upd-twist by metis
  thus ?thesis
  using  $\langle b = x \rangle$  prm-unit-action prm-unit-commutes
  using ptrm-infer-type.simps(4) ptrm-apply-prm.simps(4) by metis
next
qed
next
case (PPair  $A B$ )
  thus ?case by simp
next
case (PFst  $P$ )
  thus ?case by simp
next
case (PSnd  $P$ )
  thus ?case by simp
next
qed

lemma ptrm-infer-type-alpha-equiv:
  assumes  $X \approx Y$ 
  shows ptrm-infer-type  $\Gamma X = \text{ptrm-infer-type } \Gamma Y$ 
using assms proof(induction arbitrary:  $\Gamma$ )
  case (fn2  $a b A B T \Gamma$ )
    hence ptrm-infer-type ( $\Gamma(a \mapsto T)$ )  $A = \text{ptrm-infer-type } (\Gamma(b \mapsto T)) B$ 
    using ptrm-infer-type-swp prm-unit-commutes by metis
    thus ?case by simp
  next
qed auto

end
theory SimplyTyped
imports PreSimplyTyped
begin

quotient-type  $'a \text{ trm} = 'a \text{ ptrm} / \text{ptrm-alpha-equiv}$ 
proof(rule equivI)
  show reflp ptrm-alpha-equiv using ptrm-alpha-equiv-reflp.
  show symp ptrm-alpha-equiv using ptrm-alpha-equiv-symp.
  show transp ptrm-alpha-equiv using ptrm-alpha-equiv-transp.
qed

lift-definition Unit ::  $'a \text{ trm}$  is PUnit.

```

**lift-definition**  $Var :: 'a \Rightarrow 'a \text{ trm is } PVar.$   
**lift-definition**  $App :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm is } PApp \text{ using ptrm-alpha-equiv.app.}$   
**lift-definition**  $Fn :: 'a \Rightarrow \text{type} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm is } PFn \text{ using ptrm-alpha-equiv.fn1.}$   
**lift-definition**  $Pair :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm is } PPair \text{ using ptrm-alpha-equiv.pair.}$   
**lift-definition**  $Fst :: 'a \text{ trm} \Rightarrow 'a \text{ trm is } PFst \text{ using ptrm-alpha-equiv.fst.}$   
**lift-definition**  $Snd :: 'a \text{ trm} \Rightarrow 'a \text{ trm is } PSnd \text{ using ptrm-alpha-equiv.snd.}$   
**lift-definition**  $fvs :: 'a \text{ trm} \Rightarrow 'a \text{ set is ptrm-fvs using ptrm-alpha-equiv.fvs.}$   
**lift-definition**  $prm :: 'a \text{ prm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm (infixr } \cdot 150) \text{ is ptrm-apply-prm}$   
**using ptrm-alpha-equiv-prm.**  
**lift-definition**  $depth :: 'a \text{ trm} \Rightarrow \text{nat is size using ptrm-size-alpha-equiv.}$

**lemma**  $depth\text{-prm}:$   
**shows**  $depth (\pi \cdot A) = depth A$   
**by**( $transfer, metis \text{ ptrm-size-prm}$ )

**lemma**  $depth\text{-app}:$   
**shows**  $depth A < depth (App A B) \text{ depth } B < depth (App A B)$   
**by**( $transfer, auto$ )+

**lemma**  $depth\text{-fn}:$   
**shows**  $depth A < depth (Fn x T A)$   
**by**( $transfer, auto$ )

**lemma**  $depth\text{-pair}:$   
**shows**  $depth A < depth (Pair A B) \text{ depth } B < depth (Pair A B)$   
**by**( $transfer, auto$ )+

**lemma**  $depth\text{-fst}:$   
**shows**  $depth P < depth (Fst P)$   
**by**( $transfer, auto$ )

**lemma**  $depth\text{-snd}:$   
**shows**  $depth P < depth (Snd P)$   
**by**( $transfer, auto$ )

**lemma**  $unit\text{-not-var}:$   
**shows**  $Unit \neq Var x$   
**proof**( $transfer$ )  
**fix**  $x :: 'a$   
**show**  $\neg PUnit \approx PVar x$   
**proof**( $rule \text{ classical}$ )  
**assume**  $\neg \neg PUnit \approx PVar x$   
**hence**  $False \text{ using } unitE \text{ by } fastforce$   
**thus**  $?thesis \text{ by } blast$   
**qed**  
**qed**

**lemma**  $unit\text{-not-app}:$   
**shows**  $Unit \neq App A B$

```

proof(transfer)
  fix  $A B :: 'a\ ptrm$ 
  show  $\neg PUnit \approx PApp\ A\ B$ 
  proof(rule classical)
    assume  $\neg\neg PUnit \approx PApp\ A\ B$ 
    hence False using unitE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma unit-not-fn:
  shows  $Unit \neq Fn\ x\ T\ A$ 
proof(transfer)
  fix  $x :: 'a$  and  $T\ A$ 
  show  $\neg PUnit \approx PFn\ x\ T\ A$ 
  proof(rule classical)
    assume  $\neg\neg PUnit \approx PFn\ x\ T\ A$ 
    hence False using unitE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma unit-not-pair:
  shows  $Unit \neq PPair\ A\ B$ 
proof(transfer)
  fix  $A B :: 'a\ ptrm$ 
  show  $\neg PUnit \approx PPair\ A\ B$ 
  proof(rule classical)
    assume  $\neg\neg PUnit \approx PPair\ A\ B$ 
    hence False using unitE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma unit-not-fst:
  shows  $Unit \neq Fst\ P$ 
proof(transfer)
  fix  $P :: 'a\ ptrm$ 
  show  $\neg PUnit \approx PFst\ P$ 
  proof(rule classical)
    assume  $\neg\neg PUnit \approx PFst\ P$ 
    hence False using unitE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma unit-not-snd:
  shows  $Unit \neq Snd\ P$ 
proof(transfer)

```

```

fix P :: 'a ptrm
show  $\neg PUnit \approx PSnd P$ 
proof(rule classical)
  assume  $\neg\neg PUnit \approx PSnd P$ 
  hence False using unitE by fastforce
  thus ?thesis by blast
qed
qed

```

```

lemma var-not-app:
  shows  $Var x \neq App A B$ 
proof(transfer)
  fix x :: 'a and A B
  show  $\neg PVar x \approx PApp A B$ 
  proof(rule classical)
    assume  $\neg\neg PVar x \approx PApp A B$ 
    hence False using varE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma var-not-fn:
  shows  $Var x \neq Fn y T A$ 
proof(transfer)
  fix x y :: 'a and T A
  show  $\neg PVar x \approx PFn y T A$ 
  proof(rule classical)
    assume  $\neg\neg PVar x \approx PFn y T A$ 
    hence False using varE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma var-not-pair:
  shows  $Var x \neq Pair A B$ 
proof(transfer)
  fix x :: 'a and A B
  show  $\neg PVar x \approx PPair A B$ 
  proof(rule classical)
    assume  $\neg\neg PVar x \approx PPair A B$ 
    hence False using varE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma var-not-fst:
  shows  $Var x \neq Fst P$ 
proof(transfer)
  fix x :: 'a and P

```



```

show  $\neg PVar\ x \approx PFst\ P$ 
proof(rule classical)
  assume  $\neg\neg PVar\ x \approx PFst\ P$ 
  hence False using varE by fastforce
  thus ?thesis by blast
qed
qed

```

```

lemma var-not-snd:
  shows  $Var\ x \neq Snd\ P$ 
proof(transfer)
  fix  $x :: 'a$  and  $P$ 
  show  $\neg PVar\ x \approx PSnd\ P$ 
  proof(rule classical)
    assume  $\neg\neg PVar\ x \approx PSnd\ P$ 
    hence False using varE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma app-not-fn:
  shows  $App\ A\ B \neq Fn\ y\ T\ X$ 
proof(transfer)
  fix  $y :: 'a$  and  $A\ B\ T\ X$ 
  show  $\neg PApp\ A\ B \approx PFn\ y\ T\ X$ 
  proof(rule classical)
    assume  $\neg\neg PApp\ A\ B \approx PFn\ y\ T\ X$ 
    hence False using appE by auto
    thus ?thesis by blast
  qed
qed

```

```

lemma app-not-pair:
  shows  $App\ A\ B \neq Pair\ C\ D$ 
proof(transfer)
  fix  $A\ B\ C\ D :: 'a\ ptrm$ 
  show  $\neg PApp\ A\ B \approx PPair\ C\ D$ 
  proof(rule classical)
    assume  $\neg\neg PApp\ A\ B \approx PPair\ C\ D$ 
    hence False using appE by auto
    thus ?thesis by blast
  qed
qed

```

```

lemma app-not-fst:
  shows  $App\ A\ B \neq Fst\ P$ 
proof(transfer)
  fix  $A\ B\ P :: 'a\ ptrm$ 
  show  $\neg PApp\ A\ B \approx PFst\ P$ 

```

```

proof(rule classical)
  assume  $\neg\neg PApp\ A\ B \approx PFst\ P$ 
  hence False using appE by auto
  thus ?thesis by blast
qed
qed

```

```

lemma app-not-snd:
  shows  $App\ A\ B \neq Snd\ P$ 
proof(transfer)
  fix  $A\ B\ P :: 'a\ ptrm$ 
  show  $\neg PApp\ A\ B \approx PSnd\ P$ 
  proof(rule classical)
    assume  $\neg\neg PApp\ A\ B \approx PSnd\ P$ 
    hence False using appE by auto
    thus ?thesis by blast
  qed
qed

```

```

lemma fn-not-pair:
  shows  $Fn\ x\ T\ A \neq Pair\ C\ D$ 
proof(transfer)
  fix  $x :: 'a$  and  $T\ A\ C\ D$ 
  show  $\neg PFn\ x\ T\ A \approx PPair\ C\ D$ 
  proof(rule classical)
    assume  $\neg\neg PFn\ x\ T\ A \approx PPair\ C\ D$ 
    hence False using fnE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma fn-not-fst:
  shows  $Fn\ x\ T\ A \neq Fst\ P$ 
proof(transfer)
  fix  $x :: 'a$  and  $T\ A\ P$ 
  show  $\neg PFn\ x\ T\ A \approx PFst\ P$ 
  proof(rule classical)
    assume  $\neg\neg PFn\ x\ T\ A \approx PFst\ P$ 
    hence False using fnE by fastforce
    thus ?thesis by blast
  qed
qed

```

```

lemma fn-not-snd:
  shows  $Fn\ x\ T\ A \neq Snd\ P$ 
proof(transfer)
  fix  $x :: 'a$  and  $T\ A\ P$ 
  show  $\neg PFn\ x\ T\ A \approx PSnd\ P$ 
  proof(rule classical)

```

**assume**  $\neg\neg PFn\ x\ T\ A \approx PSnd\ P$   
**hence** *False* **using** *fnE* **by** *fastforce*  
**thus** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *pair-not-fst*:  
**shows**  $Pair\ A\ B \neq Fst\ P$   
**proof**(*transfer*)  
**fix**  $A\ B\ P :: 'a\ ptrm$   
**show**  $\neg PPair\ A\ B \approx PFst\ P$   
**proof**(*rule classical*)  
**assume**  $\neg\neg PPair\ A\ B \approx PFst\ P$   
**hence** *False* **using** *pairE* **by** *auto*  
**thus** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *pair-not-snd*:  
**shows**  $Pair\ A\ B \neq Snd\ P$   
**proof**(*transfer*)  
**fix**  $A\ B\ P :: 'a\ ptrm$   
**show**  $\neg PPair\ A\ B \approx PSnd\ P$   
**proof**(*rule classical*)  
**assume**  $\neg\neg PPair\ A\ B \approx PSnd\ P$   
**hence** *False* **using** *pairE* **by** *auto*  
**thus** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *fst-not-snd*:  
**shows**  $Fst\ P \neq Snd\ Q$   
**proof**(*transfer*)  
**fix**  $P\ Q :: 'a\ ptrm$   
**show**  $\neg PFst\ P \approx PSnd\ Q$   
**proof**(*rule classical*)  
**assume**  $\neg\neg PFst\ P \approx PSnd\ Q$   
**hence** *False* **using** *fstE* **by** *auto*  
**thus** *?thesis* **by** *blast*  
**qed**  
**qed**

**lemma** *trm-simp*:  
**shows**  
 $Var\ x = Var\ y \implies x = y$   
 $App\ A\ B = App\ C\ D \implies A = C$   
 $App\ A\ B = App\ C\ D \implies B = D$   
 $Fn\ x\ T\ A = Fn\ y\ S\ B \implies$   
 $(x = y \wedge T = S \wedge A = B) \vee (x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y])$

B)

$Pair\ A\ B = Pair\ C\ D \implies A = C$

$Pair\ A\ B = Pair\ C\ D \implies B = D$

$Fst\ P = Fst\ Q \implies P = Q$

$Snd\ P = Snd\ Q \implies P = Q$

**proof** –

**show**  $Var\ x = Var\ y \implies x = y$  **by** (*transfer*, *insert ptrm.inject varE*, *fastforce*)

**show**  $App\ A\ B = App\ C\ D \implies A = C$  **by** (*transfer*, *insert ptrm.inject appE*, *auto*)

**show**  $App\ A\ B = App\ C\ D \implies B = D$  **by** (*transfer*, *insert ptrm.inject appE*, *auto*)

**show**  $Pair\ A\ B = Pair\ C\ D \implies A = C$  **by** (*transfer*, *insert ptrm.inject pairE*, *auto*)

**show**  $Pair\ A\ B = Pair\ C\ D \implies B = D$  **by** (*transfer*, *insert ptrm.inject pairE*, *auto*)

**show**  $Fst\ P = Fst\ Q \implies P = Q$  **by** (*transfer*, *insert ptrm.inject fstE*, *auto*)

**show**  $Snd\ P = Snd\ Q \implies P = Q$  **by** (*transfer*, *insert ptrm.inject sndE*, *auto*)

**show**  $Fn\ x\ T\ A = Fn\ y\ S\ B \implies$

$(x = y \wedge T = S \wedge A = B) \vee (x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot$

$B)$

**proof**(*transfer'*)

**fix**  $x\ y :: 'a$  **and**  $T\ S :: type$  **and**  $A\ B :: 'a\ ptrm$

**assume**  $*$ :  $PFn\ x\ T\ A \approx PFn\ y\ S\ B$

**thus**  $x = y \wedge T = S \wedge A \approx B \vee x \neq y \wedge T = S \wedge x \notin ptrm-fvs\ B \wedge A \approx [x \leftrightarrow y] \cdot B$

**proof**(*induction rule: fnE[where x=x and T=T and A=A and Y=PFn y S B], metis \**)

**case** (2 C)

**thus** *?case by simp*

**next**

**case** (3 z C)

**thus** *?case by simp*

**next**

**qed**

**qed**

**qed**

**lemma** *fn-eq*:

**assumes**  $x \neq y \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot B$

**shows**  $Fn\ x\ T\ A = Fn\ y\ T\ B$

**using** *assms by (transfer', metis ptrm-alpha-equiv.fn2)*

**lemma** *trm-prm-simp*:

**shows**

$\pi \cdot Unit = Unit$

$\pi \cdot Var\ x = Var\ (\pi \$ x)$

$\pi \cdot App\ A\ B = App\ (\pi \cdot A)\ (\pi \cdot B)$

$\pi \cdot Fn\ x\ T\ A = Fn\ (\pi \$ x)\ T\ (\pi \cdot A)$

$\pi \cdot Pair\ A\ B = Pair\ (\pi \cdot A)\ (\pi \cdot B)$

```

     $\pi \cdot \text{Fst } P = \text{Fst } (\pi \cdot P)$ 
     $\pi \cdot \text{Snd } P = \text{Snd } (\pi \cdot P)$ 
apply (transfer, auto simp add: ptrm-alpha-equiv-reflexive)
apply (transfer', auto simp add: ptrm-alpha-equiv-reflexive)
apply ((transfer, auto simp add: ptrm-alpha-equiv-reflexive)+)
done

lemma trm-prm-apply-compose:
  shows  $\pi \cdot \sigma \cdot A = (\pi \diamond \sigma) \cdot A$ 
by(transfer', metis ptrm-prm-apply-compose ptrm-alpha-equiv-reflexive)

lemma fvs-finite:
  shows finite (fvs M)
by(transfer, metis ptrm-fvs-finite)

lemma fvs-simp:
  shows
    fvs Unit = {} and
    fvs (Var x) = {x}
    fvs (App A B) = fvs A  $\cup$  fvs B
    fvs (Fn x T A) = fvs A - {x}
    fvs (Pair A B) = fvs A  $\cup$  fvs B
    fvs (Fst P) = fvs P
    fvs (Snd P) = fvs P
by((transfer, simp)+)

lemma var-prm-action:
  shows  $[a \leftrightarrow b] \cdot \text{Var } a = \text{Var } b$ 
by(transfer', simp add: prm-unit-action ptrm-alpha-equiv.intros)

lemma var-prm-inaction:
  assumes  $a \neq x$   $b \neq x$ 
  shows  $[a \leftrightarrow b] \cdot \text{Var } x = \text{Var } x$ 
using assms by(transfer', simp add: prm-unit-inaction ptrm-alpha-equiv.intros)

lemma trm-prm-apply-id:
  shows  $\varepsilon \cdot M = M$ 
by(transfer', auto simp add: ptrm-prm-apply-id)

lemma trm-prm-unit-inaction:
  assumes  $a \notin \text{fvs } X$   $b \notin \text{fvs } X$ 
  shows  $[a \leftrightarrow b] \cdot X = X$ 
using assms by(transfer', metis ptrm-prm-unit-inaction)

lemma trm-prm-agreement-equiv:
  assumes  $\bigwedge a. a \in \text{ds } \pi \sigma \implies a \notin \text{fvs } M$ 
  shows  $\pi \cdot M = \sigma \cdot M$ 
using assms by(transfer, metis ptrm-prm-agreement-equiv)

```

**lemma** *trm-induct*:

**fixes**  $P :: 'a\ trm \Rightarrow bool$

**assumes**

$P\ Unit$   
 $\bigwedge x. P\ (Var\ x)$   
 $\bigwedge A\ B. \llbracket P\ A; P\ B \rrbracket \Longrightarrow P\ (App\ A\ B)$   
 $\bigwedge x\ T\ A. P\ A \Longrightarrow P\ (Fn\ x\ T\ A)$   
 $\bigwedge A\ B. \llbracket P\ A; P\ B \rrbracket \Longrightarrow P\ (Pair\ A\ B)$   
 $\bigwedge A. P\ A \Longrightarrow P\ (Fst\ A)$   
 $\bigwedge A. P\ A \Longrightarrow P\ (Snd\ A)$

**shows**  $P\ M$

**proof** –

**have**  $\bigwedge X. P\ (abs-trm\ X)$

**proof**(*rule ptrm.induct*)

**show**  $P\ (abs-trm\ PUnit)$

**using** *assms(1) Unit.abs-eq by metis*

**show**  $P\ (abs-trm\ (PVar\ x))$  **for**  $x$

**using** *assms(2) Var.abs-eq by metis*

**show**  $\llbracket P\ (abs-trm\ A); P\ (abs-trm\ B) \rrbracket \Longrightarrow P\ (abs-trm\ (PApp\ A\ B))$  **for**  $A\ B$

**using** *assms(3) App.abs-eq by metis*

**show**  $P\ (abs-trm\ A) \Longrightarrow P\ (abs-trm\ (PFn\ x\ T\ A))$  **for**  $x\ T\ A$

**using** *assms(4) Fn.abs-eq by metis*

**show**  $\llbracket P\ (abs-trm\ A); P\ (abs-trm\ B) \rrbracket \Longrightarrow P\ (abs-trm\ (PPair\ A\ B))$  **for**  $A\ B$

**using** *assms(5) Pair.abs-eq by metis*

**show**  $P\ (abs-trm\ A) \Longrightarrow P\ (abs-trm\ (PFst\ A))$  **for**  $A$

**using** *assms(6) Fst.abs-eq by metis*

**show**  $P\ (abs-trm\ A) \Longrightarrow P\ (abs-trm\ (PSnd\ A))$  **for**  $A$

**using** *assms(7) Snd.abs-eq by metis*

**qed**

**thus** *?thesis using trm.abs-induct by auto*

**qed**

**lemma** *trm-cases*:

**assumes**

$M = Unit \Longrightarrow P\ M$   
 $\bigwedge x. M = Var\ x \Longrightarrow P\ M$   
 $\bigwedge A\ B. M = App\ A\ B \Longrightarrow P\ M$   
 $\bigwedge x\ T\ A. M = Fn\ x\ T\ A \Longrightarrow P\ M$   
 $\bigwedge A\ B. M = Pair\ A\ B \Longrightarrow P\ M$   
 $\bigwedge A. M = Fst\ A \Longrightarrow P\ M$   
 $\bigwedge A. M = Snd\ A \Longrightarrow P\ M$

**shows**  $P\ M$

**using** *assms by(induction rule: trm-induct, auto)*

**lemma** *trm-depth-induct*:

**assumes**

$P\ Unit$   
 $\bigwedge x. P\ (Var\ x)$   
 $\bigwedge A\ B. \llbracket \bigwedge K. depth\ K < depth\ (App\ A\ B) \rrbracket \Longrightarrow P\ K \rrbracket \Longrightarrow P\ (App\ A\ B)$

$\bigwedge M x T A. (\bigwedge K. \text{depth } K < \text{depth } (Fn\ x\ T\ A) \implies P\ K) \implies P\ (Fn\ x\ T\ A)$   
 $\bigwedge A B. \llbracket \bigwedge K. \text{depth } K < \text{depth } (Pair\ A\ B) \implies P\ K \rrbracket \implies P\ (Pair\ A\ B)$   
 $\bigwedge A. \llbracket \bigwedge K. \text{depth } K < \text{depth } (Fst\ A) \implies P\ K \rrbracket \implies P\ (Fst\ A)$   
 $\bigwedge A. \llbracket \bigwedge K. \text{depth } K < \text{depth } (Snd\ A) \implies P\ K \rrbracket \implies P\ (Snd\ A)$   
**shows**  $P\ M$   
**proof**(*induction depth M arbitrary: M rule: less-induct*)  
**fix**  $M :: 'a\ trm$   
**assume**  $IH: \text{depth } K < \text{depth } M \implies P\ K$  **for**  $K$   
**hence**  
 $M = Unit \implies P\ M$   
 $\bigwedge x. M = Var\ x \implies P\ M$   
 $\bigwedge A B. M = App\ A\ B \implies P\ M$   
 $\bigwedge x T A. M = Fn\ x\ T\ A \implies P\ M$   
 $\bigwedge A B. M = Pair\ A\ B \implies P\ M$   
 $\bigwedge A. M = Fst\ A \implies P\ M$   
 $\bigwedge A. M = Snd\ A \implies P\ M$   
**using** *assms* **by** *blast+*  
**thus**  $P\ M$  **using** *trm-cases*[**where**  $M=M$ ] **by** *blast*  
**qed**

**context** *fresh* **begin**

**lemma** *fresh-fn*:  
**fixes**  $x :: 'a$  **and**  $S :: 'a\ set$   
**assumes** *finite S*  
**shows**  $\exists y B. y \notin S \wedge B = [y \leftrightarrow x] \cdot A \wedge (Fn\ x\ T\ A = Fn\ y\ T\ B)$   
**proof** –  
**have**  $*$ : *finite*  $(\{x\} \cup fvs\ A \cup S)$  **using** *fvs-finite* *assms* **by** *auto*  
**obtain**  $y$  **where**  $y = \text{fresh-in } (\{x\} \cup fvs\ A \cup S)$  **by** *auto*  
**hence**  $y \notin (\{x\} \cup fvs\ A \cup S)$  **using** *fresh-axioms*  $*$  **unfolding** *class.fresh-def*  
**by** *metis*  
**hence**  $y \neq x \ y \notin fvs\ A \ y \notin S$  **by** *auto*

**obtain**  $B$  **where**  $B: B = [y \leftrightarrow x] \cdot A$  **by** *auto*  
**hence**  $Fn\ x\ T\ A = Fn\ y\ T\ B$  **using** *fn-eq*  $\langle y \neq x \rangle \langle y \notin fvs\ A \rangle$  **by** *metis*  
**thus** *?thesis* **using**  $\langle y \neq x \rangle \langle y \notin S \rangle$   $B$  **by** *metis*  
**qed**

**lemma** *trm-strong-induct*:  
**fixes**  $P :: 'a\ set \implies 'a\ trm \implies bool$   
**assumes**  
 $P\ S\ Unit$   
 $\bigwedge x. P\ S\ (Var\ x)$   
 $\bigwedge A B. \llbracket P\ S\ A; P\ S\ B \rrbracket \implies P\ S\ (App\ A\ B)$   
 $\bigwedge x T. x \notin S \implies (\bigwedge A. P\ S\ A \implies P\ S\ (Fn\ x\ T\ A))$   
 $\bigwedge A B. \llbracket P\ S\ A; P\ S\ B \rrbracket \implies P\ S\ (Pair\ A\ B)$   
 $\bigwedge A. P\ S\ A \implies P\ S\ (Fst\ A)$   
 $\bigwedge A. P\ S\ A \implies P\ S\ (Snd\ A)$   
*finite S*

```

shows  $P S M$ 
proof -
  have  $\bigwedge \pi. P S (\pi \cdot M)$ 
  proof (induction  $M$  rule: trm-induct)
    case 1
      thus ?case using assms(1) trm-prm-simp(1) by metis
    next
    case (2  $x$ )
      thus ?case using assms(2) trm-prm-simp(2) by metis
    next
    case (3  $A B$ )
      thus ?case using assms(3) trm-prm-simp(3) by metis
    next
    case (4  $x T A$ )
      have finite S finite (fvs ( $\pi \cdot A$ )) finite  $\{\pi \$ x\}$ 
      using  $\langle$ finite S $\rangle$  fvs-finite by auto
      hence finite (S  $\cup$  fvs ( $\pi \cdot A$ )  $\cup$   $\{\pi \$ x\})$  by auto

      obtain  $y$  where  $y = \text{fresh-in } (S \cup \text{fvs } (\pi \cdot A) \cup \{\pi \$ x\})$  by auto
      hence  $y \notin S \cup \text{fvs } (\pi \cdot A) \cup \{\pi \$ x\}$  using fresh-axioms unfolding
      class.fresh-def
      using  $\langle$ finite (S  $\cup$  fvs ( $\pi \cdot A$ )  $\cup$   $\{\pi \$ x\})$  $\rangle$  by metis
      hence  $y \neq \pi \$ x$   $y \notin \text{fvs } (\pi \cdot A)$   $y \notin S$  by auto
      hence *:  $\bigwedge A. P S A \implies P S (Fn y T A)$  using assms(4) by metis

      have  $P S (([y \leftrightarrow \pi \$ x] \diamond \pi) \cdot A)$  using 4 by metis
      hence  $P S (Fn y T (([y \leftrightarrow \pi \$ x] \diamond \pi) \cdot A))$  using * by metis
      moreover have  $(Fn y T (([y \leftrightarrow \pi \$ x] \diamond \pi) \cdot A)) = Fn (\pi \$ x) T (\pi \cdot A)$ 
      using trm-prm-apply-compose fn-eq  $\langle$  $y \neq \pi \$ x$  $\rangle$   $\langle$  $y \notin \text{fvs } (\pi \cdot A)$  $\rangle$  by metis
      ultimately show ?case using trm-prm-simp(4) by metis
    next
    case (5  $A B$ )
      thus ?case using assms(5) trm-prm-simp(5) by metis
    next
    case (6  $A$ )
      thus ?case using assms(6) trm-prm-simp(6) by metis
    next
    case (7  $A$ )
      thus ?case using assms(7) trm-prm-simp(7) by metis
    next
  qed
  hence  $P S (\varepsilon \cdot M)$  by metis
  thus  $P S M$  using trm-prm-apply-id by metis
qed

```

```

lemma trm-strong-cases:
  fixes  $P :: 'a \text{ set} \Rightarrow 'a \text{ trm} \Rightarrow \text{bool}$ 
  assumes
     $M = \text{Unit} \implies P S M$ 

```



$\wedge x. M = \text{Var } x \implies P S M$   
 $\wedge A B. M = \text{App } A B \implies P S M$   
 $\wedge x T A. M = \text{Fn } x T A \implies x \notin S \implies P S M$   
 $\wedge A B. M = \text{Pair } A B \implies P S M$   
 $\wedge A. M = \text{Fst } A \implies P S M$   
 $\wedge A. M = \text{Snd } A \implies P S M$   
*finite S*  
**shows**  $P S M$   
**using** *assms* **by**(*induction S M rule: trm-strong-induct, metis+*)

**lemma** *trm-strong-depth-induct*:  
**fixes**  $P :: 'a \text{ set} \Rightarrow 'a \text{ trm} \Rightarrow \text{bool}$   
**assumes**  
 $P S \text{Unit}$   
 $\wedge x. P S (\text{Var } x)$   
 $\wedge A B. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{App } A B) \implies P S K \rrbracket \implies P S (\text{App } A B)$   
 $\wedge x T. x \notin S \implies (\wedge A. (\wedge K. \text{depth } K < \text{depth } (\text{Fn } x T A) \implies P S K) \implies P S (\text{Fn } x T A))$   
 $\wedge A B. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Pair } A B) \implies P S K \rrbracket \implies P S (\text{Pair } A B)$   
 $\wedge A. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Fst } A) \implies P S K \rrbracket \implies P S (\text{Fst } A)$   
 $\wedge A. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Snd } A) \implies P S K \rrbracket \implies P S (\text{Snd } A)$   
*finite S*  
**shows**  $P S M$   
**proof**(*induction depth M arbitrary: M rule: less-induct*)  
**fix**  $M :: 'a \text{ trm}$   
**assume** *IH: depth K < depth M  $\implies P S K$  for K*  
**hence**  
 $M = \text{Unit} \implies P S M$   
 $\wedge x. M = \text{Var } x \implies P S M$   
 $\wedge A B. M = \text{App } A B \implies P S M$   
 $\wedge x T A. M = \text{Fn } x T A \implies x \notin S \implies P S M$   
 $\wedge A B. M = \text{Pair } A B \implies P S M$   
 $\wedge A. M = \text{Fst } A \implies P S M$   
 $\wedge A. M = \text{Snd } A \implies P S M$   
*finite S*  
**using** *assms IH* **by** *metis+*  
**thus**  $P S M$  **using** *trm-strong-cases[where M=M]* **by** *blast*  
**qed**

**lemma** *trm-prm-fvs*:  
**shows**  $\text{fvs } (\pi \cdot M) = \pi \{ \$ \} \text{fvs } M$   
**by**(*transfer, metis ptrm-prm-fvs*)

**inductive** *typing*  $:: 'a \text{ typing-ctx} \Rightarrow 'a \text{ trm} \Rightarrow \text{type} \Rightarrow \text{bool} \ (- \vdash - : -)$  **where**  
 $tunit: \Gamma \vdash \text{Unit} : TUnit$   
 $| tvar: \Gamma x = \text{Some } \tau \implies \Gamma \vdash \text{Var } x : \tau$   
 $| tapp: \llbracket \Gamma \vdash f : (TArr \tau \sigma); \Gamma \vdash x : \tau \rrbracket \implies \Gamma \vdash \text{App } f x : \sigma$   
 $| tfn: \Gamma(x \mapsto \tau) \vdash A : \sigma \implies \Gamma \vdash \text{Fn } x \tau A : (TArr \tau \sigma)$   
 $| tpair: \llbracket \Gamma \vdash A : \tau; \Gamma \vdash B : \sigma \rrbracket \implies \Gamma \vdash \text{Pair } A B : (TPair \tau \sigma)$

| *tfst*:  $\Gamma \vdash P : (TPair \tau \sigma) \implies \Gamma \vdash Fst P : \tau$   
| *tsnd*:  $\Gamma \vdash P : (TPair \tau \sigma) \implies \Gamma \vdash Snd P : \sigma$

**lemma** *typing-prm*:

**assumes**  $\Gamma \vdash M : \tau \wedge y. y \in fvs M \implies \Gamma y = \Delta (\pi \$ y)$

**shows**  $\Delta \vdash \pi \cdot M : \tau$

**using** *assms* **proof**(*induction arbitrary*:  $\Delta$  *rule*: *typing.induct*)

**case** (*tunit*  $\Gamma$ )

**thus** *?case* **using** *typing.tunit trm-prm-simp(1)* **by** *metis*

**next**

**case** (*tvar*  $\Gamma x \tau$ )

**thus** *?case* **using** *typing.tvar trm-prm-simp(2) fvs-simp(2) singletonI* **by** *metis*

**next**

**case** (*tapp*  $\Gamma A \tau \sigma B$ )

**thus** *?case* **using** *typing.tapp trm-prm-simp(3) fvs-simp(3) UnCI* **by** *metis*

**next**

**case** (*tfn*  $\Gamma x \tau A \sigma$ )

**have**  $y \in fvs A \implies (\Gamma(x \mapsto \tau)) y = (\Delta(\pi \$ x \mapsto \tau)) (\pi \$ y)$  **for**  $y$

**proof**(*cases*  $y = x$ )

**case** *True*

**thus** *?thesis* **using** *fun-upd-apply* **by** *simp*

**next**

**case** *False*

**assume**  $y \in fvs A$

**hence**  $y \in fvs (Fn x \tau A)$  **using** *fvs-simp(4)*  $\langle y \neq x \rangle$  *DiffI singletonD* **by**

*fastforce*

**hence**  $\Gamma y = \Delta (\pi \$ y)$  **using** *tfn.prem*s **by** *metis*

**thus** *?thesis* **by** (*simp add*: *prm-apply-unequal*)

**next**

**qed**

**hence**  $\Delta(\pi \$ x \mapsto \tau) \vdash \pi \cdot A : \sigma$  **using** *tfn.IH* **by** *metis*

**thus** *?case* **using** *trm-prm-simp(4) typing.tfn* **by** *metis*

**next**

**case** (*tpair*  $\Gamma A B$ )

**thus** *?case* **using** *typing.tpair trm-prm-simp(5) fvs-simp(5) UnCI* **by** *metis*

**next**

**case** (*tfst*  $\Gamma P \tau \sigma$ )

**thus** *?case* **using** *typing.tfst trm-prm-simp(6) fvs-simp(6)* **by** *metis*

**next**

**case** (*tsnd*  $\Gamma P \tau \sigma$ )

**thus** *?case* **using** *typing.tsnd trm-prm-simp(7) fvs-simp(7)* **by** *metis*

**next**

**qed**

**lemma** *typing-swp*:

**assumes**  $\Gamma(a \mapsto \sigma) \vdash M : \tau \ b \notin fvs M$

**shows**  $\Gamma(b \mapsto \sigma) \vdash [a \leftrightarrow b] \cdot M : \tau$

**proof** –

**have**  $y \in fvs M \implies (\Gamma(a \mapsto \sigma)) y = (\Gamma(b \mapsto \sigma)) ([a \leftrightarrow b] \$ y)$  **for**  $y$

```

proof –
  assume  $y \in fvs\ M$ 
  hence  $y \neq b$  using assms(2) by auto
  consider  $y = a \mid y \neq a$  by auto
  thus  $(\Gamma(a \mapsto \sigma))\ y = (\Gamma(b \mapsto \sigma))\ ([a \leftrightarrow b]\ \$\ y)$ 
  by(cases, simp add: prm-unit-action, simp add: prm-unit-inaction (y ≠ b))
qed
thus ?thesis using typing-prm assms(1) by metis
qed

lemma typing-unitE:
  assumes  $\Gamma \vdash Unit : \tau$ 
  shows  $\tau = TUnit$ 
using assms
  apply cases
  apply blast
  apply (auto simp add: unit-not-var unit-not-app unit-not-fn unit-not-pair unit-not-fst
unit-not-snd)
done

lemma typing-varE:
  assumes  $\Gamma \vdash Var\ x : \tau$ 
  shows  $\Gamma\ x = Some\ \tau$ 
using assms
  apply cases
  prefer 2
  apply (metis trm-simp(1))
  apply (metis unit-not-var)
  apply (auto simp add: var-not-app var-not-fn var-not-pair var-not-fst var-not-snd)
done

lemma typing-appE:
  assumes  $\Gamma \vdash App\ A\ B : \sigma$ 
  shows  $\exists \tau. (\Gamma \vdash A : (TArr\ \tau\ \sigma)) \wedge (\Gamma \vdash B : \tau)$ 
using assms
  apply cases
  prefer 3
  apply (metis trm-simp(2, 3))
  apply (metis unit-not-app)
  apply (metis var-not-app)
  apply (auto simp add: app-not-fn app-not-pair app-not-fst app-not-snd)
done

lemma typing-fnE:
  assumes  $\Gamma \vdash Fn\ x\ T\ A : \vartheta$ 
  shows  $\exists \sigma. \vartheta = (TArr\ T\ \sigma) \wedge (\Gamma(x \mapsto T) \vdash A : \sigma)$ 
using assms proof(cases)
  case (tfn y S B σ)
  from this consider

```

```

     $x = y \wedge T = S \wedge A = B \mid x \neq y \wedge T = S \wedge x \notin \text{fv} B \wedge A = [x \leftrightarrow y] \cdot B$ 
    using trm-simp(4) by metis
  thus ?thesis proof(cases)
    case 1
      thus ?thesis using tfn by metis
    next
      case 2
        thus ?thesis using tfn typing-swp prm-unit-commutes by metis
      next
        qed
    next
  qed (
    metis unit-not-fn,
    metis var-not-fn,
    metis app-not-fn,
    metis fn-not-pair,
    metis fn-not-fst,
    metis fn-not-snd
  )

```

```

lemma typing-pairE:
  assumes  $\Gamma \vdash \text{Pair } A \ B : \vartheta$ 
  shows  $\exists \tau \ \sigma. \ \vartheta = (\text{TPair } \tau \ \sigma) \wedge (\Gamma \vdash A : \tau) \wedge (\Gamma \vdash B : \sigma)$ 
using assms proof(cases)
  case (tpair  $A \ \tau \ B \ \sigma$ )
    thus ?thesis using trm-simp(5) trm-simp(6) by blast
  next
  qed (
    metis unit-not-pair,
    metis var-not-pair,
    metis app-not-pair,
    metis fn-not-pair,
    metis pair-not-fst,
    metis pair-not-snd
  )

```

```

lemma typing-fstE:
  assumes  $\Gamma \vdash \text{Fst } P : \tau$ 
  shows  $\exists \sigma. (\Gamma \vdash P : (\text{TPair } \tau \ \sigma))$ 
using assms proof(cases)
  case (tfst  $P \ \sigma$ )
    thus ?thesis using trm-simp(7) by blast
  next
  qed (
    metis unit-not-fst,
    metis var-not-fst,
    metis app-not-fst,
    metis fn-not-fst,
    metis pair-not-fst,
  )

```

*metis fst-not-snd*  
 )

**lemma** *typing-sndE*:  
 assumes  $\Gamma \vdash \text{Snd } P : \sigma$   
 shows  $\exists \tau. (\Gamma \vdash P : (\text{TPair } \tau \ \sigma))$   
**using** *assms proof(cases)*  
 case (*tsnd*  $P \ \sigma$ )  
 thus *?thesis using trm-simp(8) by blast*  
**next**  
**qed** (  
*metis unit-not-snd,*  
*metis var-not-snd,*  
*metis app-not-snd,*  
*metis fn-not-snd,*  
*metis pair-not-snd,*  
*metis fst-not-snd*  
 )

**theorem** *typing-weaken*:  
 assumes  $\Gamma \vdash M : \tau \ y \notin \text{fvs } M$   
 shows  $\Gamma(y \mapsto \sigma) \vdash M : \tau$   
**using** *assms proof(induction rule: typing.induct)*  
 case (*tunit*  $\Gamma$ )  
 thus *?case using typing.tunit by metis*  
**next**  
 case (*tvar*  $\Gamma \ x \ \tau$ )  
 hence  $y \neq x$  **using** *fvs-simp(2) singletonI by force*  
 hence  $(\Gamma(y \mapsto \sigma)) \ x = \text{Some } \tau$  **using** *tvar.hyps fun-upd-apply by simp*  
 thus *?case using typing.tvar by metis*  
**next**  
 case (*tapp*  $\Gamma \ f \ \tau \ \tau' \ x$ )  
 from  $\langle y \notin \text{fvs } (\text{App } f \ x) \rangle$  **have**  $y \notin \text{fvs } f \ y \notin \text{fvs } x$  **using** *fvs-simp(3) Un-iff*  
**by force+**  
 hence  $\Gamma(y \mapsto \sigma) \vdash f : (\text{TArr } \tau \ \tau') \ \Gamma(y \mapsto \sigma) \vdash x : \tau$  **using** *tapp.IH by metis+*  
 thus *?case using typing.tapp by metis*  
**next**  
 case (*tfn*  $\Gamma \ x \ \tau \ A \ \tau'$ )  
 from  $\langle y \notin \text{fvs } (\text{Fn } x \ \tau \ A) \rangle$  **consider**  $y = x \mid y \neq x \wedge y \notin \text{fvs } A$   
**using** *fvs-simp(4) DiffI empty-iff insert-iff by fastforce*  
 thus *?case proof(cases)*  
 case 1  
 hence  $(\Gamma(y \mapsto \sigma)(x \mapsto \tau)) \vdash A : \tau'$  **using** *tfn.hyps fun-upd-upd by simp*  
 thus *?thesis using typing.tfn by metis*  
**next**  
 case 2  
 hence  $y \neq x \ y \notin \text{fvs } A$  **by auto**  
 hence  $\Gamma(x \mapsto \tau, y \mapsto \sigma) \vdash A : \tau'$  **using** *tfn.IH by metis*  
 hence  $\Gamma(y \mapsto \sigma, x \mapsto \tau) \vdash A : \tau'$  **using**  $\langle y \neq x \rangle$  *fun-upd-twist by metis*

```

      thus ?thesis using typing.tfn by metis
    next
  qed
next
case (tpair  $\Gamma A \tau B \sigma$ )
  thus ?case using typing.tpair fvs-simp(5) UnCI by metis
next
case (tfst  $\Gamma P \tau \sigma$ )
  thus ?case using typing.tfst fvs-simp(6) by metis
next
case (tsnd  $\Gamma P \tau \sigma$ )
  thus ?case using typing.tsnd fvs-simp(7) by metis
next
qed

```

**lift-definition** *infer* :: 'a typing-ctx  $\Rightarrow$  'a trm  $\Rightarrow$  type option is ptrm-infer-type  
**using** ptrm-infer-type-alpha-equiv.

**export-code** *infer* fresh-nat-inst.fresh-in-nat **in** Haskell

**lemma** *infer-simp*:

**shows**

```

infer  $\Gamma$  Unit = Some TUnit
infer  $\Gamma$  (Var x) =  $\Gamma$  x
infer  $\Gamma$  (App A B) = (case (infer  $\Gamma$  A, infer  $\Gamma$  B) of
  (Some (TArr  $\tau \sigma$ ), Some  $\tau'$ )  $\Rightarrow$  (if  $\tau = \tau'$  then Some  $\sigma$  else None)
  | -  $\Rightarrow$  None
)
infer  $\Gamma$  (Fn x  $\tau$  A) = (case infer ( $\Gamma(x \mapsto \tau)$ ) A of
  Some  $\sigma \Rightarrow$  Some (TArr  $\tau \sigma$ )
  | None  $\Rightarrow$  None
)
infer  $\Gamma$  (Pair A B) = (case (infer  $\Gamma$  A, infer  $\Gamma$  B) of
  (Some  $\tau$ , Some  $\sigma$ )  $\Rightarrow$  Some (TPair  $\tau \sigma$ )
  | -  $\Rightarrow$  None
)
infer  $\Gamma$  (Fst P) = (case infer  $\Gamma$  P of
  (Some (TPair  $\tau \sigma$ ))  $\Rightarrow$  Some  $\tau$ 
  | -  $\Rightarrow$  None
)
infer  $\Gamma$  (Snd P) = (case infer  $\Gamma$  P of
  (Some (TPair  $\tau \sigma$ ))  $\Rightarrow$  Some  $\sigma$ 
  | -  $\Rightarrow$  None
)

```

**by**((transfer, simp)+)

**lemma** *infer-unitE*:

**assumes** *infer*  $\Gamma$  Unit = Some  $\tau$

**shows**  $\tau = TUnit$   
**using** *assms* **by**(*transfer*, *simp*)

**lemma** *infer-varE*:  
**assumes** *infer*  $\Gamma (Var\ x) = Some\ \tau$   
**shows**  $\Gamma\ x = Some\ \tau$   
**using** *assms* **by**(*transfer*, *simp*)

**lemma** *infer-appE*:  
**assumes** *infer*  $\Gamma (App\ A\ B) = Some\ \tau$   
**shows**  $\exists\sigma. infer\ \Gamma\ A = Some\ (TArr\ \sigma\ \tau) \wedge infer\ \Gamma\ B = Some\ \sigma$   
**using** *assms* **proof**(*transfer*)  
**fix**  $\Gamma :: 'a\ typing\ ctx$  **and**  $A\ B\ \tau$   
**assume**  $H: ptrm\ infer\ type\ \Gamma (PApp\ A\ B) = Some\ \tau$

**have** *ptrm-infer-type*  $\Gamma\ A \neq None$   
**proof**(*rule classical*, *auto*)  
**assume** *ptrm-infer-type*  $\Gamma\ A = None$   
**hence** *ptrm-infer-type*  $\Gamma (PApp\ A\ B) = None$  **by** *auto*  
**thus** *False* **using**  $H$  **by** *auto*

**qed**  
**from** *this* **obtain**  $T$  **where**  $*$ : *ptrm-infer-type*  $\Gamma\ A = Some\ T$  **by** *auto*

**have**  $T \neq TVar\ x$  **for**  $x$   
**proof**(*rule classical*, *auto*)  
**fix**  $x$   
**assume**  $T = TVar\ x$   
**hence** *ptrm-infer-type*  $\Gamma\ A = Some\ (TVar\ x)$  **using**  $*$  **by** *metis*  
**hence** *ptrm-infer-type*  $\Gamma (PApp\ A\ B) = None$  **by** *simp*  
**thus** *False* **using**  $H$  **by** *auto*

**qed**  
**moreover** **have**  $T \neq TUnit$   
**proof**(*rule classical*, *auto*)  
**fix**  $x$   
**assume**  $T = TUnit$   
**hence** *ptrm-infer-type*  $\Gamma\ A = Some\ TUnit$  **using**  $*$  **by** *metis*  
**hence** *ptrm-infer-type*  $\Gamma (PApp\ A\ B) = None$  **by** *simp*  
**thus** *False* **using**  $H$  **by** *auto*

**qed**  
**moreover** **have**  $T \neq TPair\ \tau\ \sigma$  **for**  $\tau\ \sigma$   
**proof**(*rule classical*, *auto*)  
**fix**  $\tau\ \sigma$   
**assume**  $T = TPair\ \tau\ \sigma$   
**hence** *ptrm-infer-type*  $\Gamma\ A = Some\ (TPair\ \tau\ \sigma)$  **using**  $*$  **by** *metis*  
**hence** *ptrm-infer-type*  $\Gamma (PApp\ A\ B) = None$  **by** *simp*  
**thus** *False* **using**  $H$  **by** *auto*

**qed**  
**ultimately** **obtain**  $\sigma\ \tau'$  **where**  $T = TArr\ \sigma\ \tau'$  **by**(*cases*  $T$ , *blast*, *auto*)  
**hence**  $*$ : *ptrm-infer-type*  $\Gamma\ A = Some\ (TArr\ \sigma\ \tau')$  **using**  $*$  **by** *metis*

```

have ptrm-infer-type  $\Gamma B \neq \text{None}$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma B = \text{None}$ 
  hence ptrm-infer-type  $\Gamma (PApp A B) = \text{None}$  using * by auto
  thus False using H by auto
qed
from this obtain  $\sigma'$  where **: ptrm-infer-type  $\Gamma B = \text{Some } \sigma'$  by auto

have  $\sigma = \sigma'$ 
proof(rule classical)
  assume  $\sigma \neq \sigma'$ 
  hence ptrm-infer-type  $\Gamma (PApp A B) = \text{None}$  using * ** by simp
  hence False using H by auto
  thus  $\sigma = \sigma'$  by blast
qed
hence **: ptrm-infer-type  $\Gamma B = \text{Some } \sigma$  using ** by auto

have ptrm-infer-type  $\Gamma (PApp A B) = \text{Some } \tau'$  using * ** by auto
hence  $\tau = \tau'$  using H by auto
hence *: ptrm-infer-type  $\Gamma A = \text{Some } (TArr \sigma \tau)$  using * by auto

show  $\exists \sigma. \text{ptrm-infer-type } \Gamma A = \text{Some } (TArr \sigma \tau) \wedge \text{ptrm-infer-type } \Gamma B =$ 
Some  $\sigma$ 
using * ** by auto
qed

lemma infer-fnE:
assumes infer  $\Gamma (Fn x T A) = \text{Some } \tau$ 
shows  $\exists \sigma. \tau = TArr T \sigma \wedge \text{infer } (\Gamma(x \mapsto T)) A = \text{Some } \sigma$ 
using assms proof(transfer)
fix  $x :: 'a$  and  $\Gamma T A \tau$ 
assume H: ptrm-infer-type  $\Gamma (PFn x T A) = \text{Some } \tau$ 

have ptrm-infer-type  $(\Gamma(x \mapsto T)) A \neq \text{None}$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $(\Gamma(x \mapsto T)) A = \text{None}$ 
  hence ptrm-infer-type  $\Gamma (PFn x T A) = \text{None}$  by auto
  thus False using H by auto
qed
from this obtain  $\sigma$  where *: ptrm-infer-type  $(\Gamma(x \mapsto T)) A = \text{Some } \sigma$  by auto

have ptrm-infer-type  $\Gamma (PFn x T A) = \text{Some } (TArr T \sigma)$  using * by auto
hence  $\tau = TArr T \sigma$  using H by auto
thus  $\exists \sigma. \tau = TArr T \sigma \wedge \text{ptrm-infer-type } (\Gamma(x \mapsto T)) A = \text{Some } \sigma$ 
using * by auto
qed

lemma infer-pairE:

```



```

assumes infer  $\Gamma$  (Pair A B) = Some  $\tau$ 
shows  $\exists T S. \tau = TPair T S \wedge infer \Gamma A = Some T \wedge infer \Gamma B = Some S$ 
using assms proof(transfer)
fix A B :: 'a ptrm and  $\Gamma \tau$ 
assume H: ptrm-infer-type  $\Gamma$  (PPair A B) = Some  $\tau$ 

have ptrm-infer-type  $\Gamma A \neq None$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma A = None$ 
  hence ptrm-infer-type  $\Gamma$  (PPair A B) = None by auto
  thus False using H by auto
qed
moreover have ptrm-infer-type  $\Gamma B \neq None$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma B = None$ 
  hence ptrm-infer-type  $\Gamma$  (PPair A B) = None by (simp add: option.case-eq-if)
  thus False using H by auto
qed
ultimately obtain T S
  where  $\tau = TPair T S$  ptrm-infer-type  $\Gamma A = Some T$  ptrm-infer-type  $\Gamma B =$ 
  Some S
  using H by auto
  thus  $\exists T S. \tau = TPair T S \wedge ptrm-infer-type \Gamma A = Some T \wedge ptrm-infer-type$ 
 $\Gamma B = Some S$  by auto
qed

lemma infer-fstE:
assumes infer  $\Gamma$  (Fst P) = Some  $\tau$ 
shows  $\exists T S. infer \Gamma P = Some (TPair T S) \wedge \tau = T$ 
using assms proof(transfer)
fix P :: 'a ptrm and  $\Gamma \tau$ 
assume H: ptrm-infer-type  $\Gamma$  (PFst P) = Some  $\tau$ 

have ptrm-infer-type  $\Gamma P \neq None$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma P = None$ 
  thus False using H by simp
qed
moreover have ptrm-infer-type  $\Gamma P \neq Some TUnit$ 
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma P = Some TUnit$ 
  thus False using H by simp
qed
moreover have ptrm-infer-type  $\Gamma P \neq Some (TVar x)$  for x
proof(rule classical, auto)
  assume ptrm-infer-type  $\Gamma P = Some (TVar x)$ 
  thus False using H by simp
qed
moreover have ptrm-infer-type  $\Gamma P \neq Some (TArr T S)$  for T S

```

**proof**(*rule classical, auto*)  
 assume *ptrm-infer-type*  $\Gamma P = \text{Some } (TArr T S)$   
 thus *False* using *H* by *simp*  
**qed**  
 ultimately obtain *T S* where  
   *ptrm-infer-type*  $\Gamma P = \text{Some } (TPair T S)$   
   using *type.distinct type.exhaust option.exhaust* by *metis*  
 moreover hence *ptrm-infer-type*  $\Gamma (PFst P) = \text{Some } T$  by *simp*  
 ultimately show  $\exists T S. \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S) \wedge \tau = T$   
 using *H* by *auto*  
**qed**

**lemma** *infer-sndE*:  
 assumes *infer*  $\Gamma (Snd P) = \text{Some } \tau$   
 shows  $\exists T S. \text{infer } \Gamma P = \text{Some } (TPair T S) \wedge \tau = S$   
**using** *assms* **proof**(*transfer*)  
 fix *P* :: 'a *ptrm* and  $\Gamma \tau$   
 assume *H*: *ptrm-infer-type*  $\Gamma (PSnd P) = \text{Some } \tau$

have *ptrm-infer-type*  $\Gamma P \neq \text{None}$   
**proof**(*rule classical, auto*)  
 assume *ptrm-infer-type*  $\Gamma P = \text{None}$   
 thus *False* using *H* by *simp*  
**qed**

moreover have *ptrm-infer-type*  $\Gamma P \neq \text{Some } TUnit$   
**proof**(*rule classical, auto*)  
 assume *ptrm-infer-type*  $\Gamma P = \text{Some } TUnit$   
 thus *False* using *H* by *simp*  
**qed**

moreover have *ptrm-infer-type*  $\Gamma P \neq \text{Some } (TVar x)$  for *x*  
**proof**(*rule classical, auto*)  
 assume *ptrm-infer-type*  $\Gamma P = \text{Some } (TVar x)$   
 thus *False* using *H* by *simp*  
**qed**

moreover have *ptrm-infer-type*  $\Gamma P \neq \text{Some } (TArr T S)$  for *T S*  
**proof**(*rule classical, auto*)  
 assume *ptrm-infer-type*  $\Gamma P = \text{Some } (TArr T S)$   
 thus *False* using *H* by *simp*  
**qed**

ultimately obtain *T S* where  
   *ptrm-infer-type*  $\Gamma P = \text{Some } (TPair T S)$   
   using *type.distinct type.exhaust option.exhaust* by *metis*  
 moreover hence *ptrm-infer-type*  $\Gamma (PSnd P) = \text{Some } S$  by *simp*  
 ultimately show  $\exists T S. \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S) \wedge \tau = S$   
 using *H* by *auto*  
**qed**

**lemma** *infer-sound*:  
 assumes *infer*  $\Gamma M = \text{Some } \tau$

```

shows  $\Gamma \vdash M : \tau$ 
using assms proof(induction M arbitrary:  $\Gamma \tau$  rule: trm-induct)
case 1
  thus ?case using infer-unitE typing.tunit by metis
next
case (2 x)
  hence  $\Gamma x = \text{Some } \tau$  using infer-varE by metis
  thus ?case using typing.tvar by metis
next
case (3 A B)
  from  $\langle \text{infer } \Gamma (App\ A\ B) = \text{Some } \tau \rangle$  obtain  $\sigma$ 
  where  $\text{infer } \Gamma A = \text{Some } (TArr\ \sigma\ \tau)$  and  $\text{infer } \Gamma B = \text{Some } \sigma$ 
  using infer-appE by metis
  thus ?case using 3.IH typing.tapp by metis
next
case (4 x T A  $\Gamma \tau$ )
  from  $\langle \text{infer } \Gamma (Fn\ x\ T\ A) = \text{Some } \tau \rangle$  obtain  $\sigma$ 
  where  $\tau = TArr\ T\ \sigma$  and  $\text{infer } (\Gamma(x \mapsto T))\ A = \text{Some } \sigma$ 
  using infer-fnE by metis
  thus ?case using 4.IH typing.tfn by metis
next
case (5 A B  $\Gamma \tau$ )
  thus ?case using typing.tpair infer-pairE by metis
next
case (6 P  $\Gamma \tau$ )
  thus ?case using typing.tfst infer-fstE by metis
next
case (7 P  $\Gamma \tau$ )
  thus ?case using typing.tsnd infer-sndE by metis
next
qed

```

```

lemma infer-complete:
  assumes  $\Gamma \vdash M : \tau$ 
  shows  $\text{infer } \Gamma M = \text{Some } \tau$ 
using assms proof(induction)
  case (tfn  $\Gamma x \tau A \sigma$ )
    thus ?case by (simp add: infer-simp(4) tfn.IH)
  next
qed (auto simp add: infer-simp)

```

```

theorem infer-valid:
  shows  $(\Gamma \vdash M : \tau) = (\text{infer } \Gamma M = \text{Some } \tau)$ 
using infer-sound infer-complete by blast

```

```

inductive substitutes :: 'a trm  $\Rightarrow$  'a  $\Rightarrow$  'a trm  $\Rightarrow$  'a trm  $\Rightarrow$  bool where
  unit: substitutes Unit y M Unit
| var1:  $x = y \Longrightarrow \text{substitutes } (Var\ x)\ y\ M\ M$ 
| var2:  $x \neq y \Longrightarrow \text{substitutes } (Var\ x)\ y\ M\ (Var\ x)$ 

```

| *app*:  $\llbracket \text{substitutes } A \ x \ M \ A'; \text{ substitutes } B \ x \ M \ B' \rrbracket \implies \text{substitutes } (\text{App } A \ B) \ x \ M \ (\text{App } A' \ B')$   
| *fn*:  $\llbracket x \neq y; y \notin \text{fvs } M; \text{ substitutes } A \ x \ M \ A' \rrbracket \implies \text{substitutes } (\text{Fn } y \ T \ A) \ x \ M \ (\text{Fn } y \ T \ A')$   
| *pair*:  $\llbracket \text{substitutes } A \ x \ M \ A'; \text{ substitutes } B \ x \ M \ B' \rrbracket \implies \text{substitutes } (\text{Pair } A \ B) \ x \ M \ (\text{Pair } A' \ B')$   
| *fst*:  $\text{substitutes } P \ x \ M \ P' \implies \text{substitutes } (\text{Fst } P) \ x \ M \ (\text{Fst } P')$   
| *snd*:  $\text{substitutes } P \ x \ M \ P' \implies \text{substitutes } (\text{Snd } P) \ x \ M \ (\text{Snd } P')$

**lemma** *substitutes-prm*:

**assumes** *substitutes*  $A \ x \ M \ A'$   
**shows** *substitutes*  $(\pi \cdot A) \ (\pi \$ x) \ (\pi \cdot M) \ (\pi \cdot A')$   
**using** *assms* **proof**(*induction*)  
  **case** (*unit*  $y \ M$ )  
    **thus** ?*case* **using** *substitutes.unit trm-prm-simp(1)* **by** *metis*  
  **case** (*var1*  $x \ y \ M$ )  
    **thus** ?*case* **using** *substitutes.var1 trm-prm-simp(2)* **by** *metis*  
  **next**  
  **case** (*var2*  $x \ y \ M$ )  
    **thus** ?*case* **using** *substitutes.var2 trm-prm-simp(2) prm-apply-unequal* **by** *metis*  
  **next**  
  **case** (*app*  $A \ x \ M \ A' \ B \ B'$ )  
    **thus** ?*case* **using** *substitutes.app trm-prm-simp(3)* **by** *metis*  
  **next**  
  **case** (*fn*  $x \ y \ M \ A \ A' \ T$ )  
    **thus** ?*case*  
    **using** *substitutes.fn trm-prm-simp(4) prm-apply-unequal prm-set-notmembership trm-prm-fvs*  
    **by** *metis*  
  **next**  
  **case** (*pair*  $A \ x \ M \ A' \ B \ B'$ )  
    **thus** ?*case* **using** *substitutes.pair trm-prm-simp(5)* **by** *metis*  
  **next**  
  **case** (*fst*  $P \ x \ M \ P'$ )  
    **thus** ?*case* **using** *substitutes.fst trm-prm-simp(6)* **by** *metis*  
  **next**  
  **case** (*snd*  $P \ x \ M \ P'$ )  
    **thus** ?*case* **using** *substitutes.snd trm-prm-simp(7)* **by** *metis*  
  **next**  
**qed**

**lemma** *substitutes-fvs*:

**assumes** *substitutes*  $A \ x \ M \ A'$   
**shows**  $\text{fvs } A' \subseteq \text{fvs } A - \{x\} \cup \text{fvs } M$   
**using** *assms* **proof**(*induction*)  
  **case** (*unit*  $y \ M$ )  
    **thus** ?*case* **using** *fvs-simp(1)* **by** *auto*  
  **case** (*var1*  $x \ y \ M$ )  
    **thus** ?*case* **by** *auto*

```

next
case (var2 x y M)
  thus ?case
    using fvs-simp(2) Un-subset-iff Un-upper2 insert-Diff-if insert-is-Un single-
tonD sup-commute
    by metis
next
case (app A x M A' B B')
  hence fvs A' ∪ fvs B' ⊆ (fvs A - {x} ∪ fvs M) ∪ (fvs B - {x} ∪ fvs M) by
auto
  hence fvs A' ∪ fvs B' ⊆ (fvs A ∪ fvs B) - {x} ∪ fvs M by auto
  thus ?case using fvs-simp(3) by metis
next
case (fn x y M A A' T)
  hence fvs A' - {y} ⊆ fvs A - {y} - {x} ∪ fvs M by auto
  thus ?case using fvs-simp(4) by metis
next
case (pair A x M A' B B')
  hence fvs A' ∪ fvs B' ⊆ (fvs A - {x} ∪ fvs M) ∪ (fvs B - {x} ∪ fvs M) by
auto
  hence fvs A' ∪ fvs B' ⊆ (fvs A ∪ fvs B) - {x} ∪ fvs M by auto
  thus ?case using fvs-simp(5) by metis
next
case (fst P x M P')
  thus ?case using fvs-simp(6) by fastforce
next
case (snd P x M P')
  thus ?case using fvs-simp(7) by fastforce
next
qed

```

**inductive-cases** *substitutes-unitE'*: *substitutes Unit y M X*

**lemma** *substitutes-unitE*:

**assumes** *substitutes Unit y M X*

**shows**  $X = \text{Unit}$

**by**(

*rule* *substitutes-unitE'*[**where**  $y=y$  **and**  $M=M$  **and**  $X=X$ ],

*metis* *assms*,

*auto simp add: unit-not-var unit-not-app unit-not-fn unit-not-pair unit-not-fst*  
*unit-not-snd*  
)

**inductive-cases** *substitutes-varE'*: *substitutes (Var x) y M X*

**lemma** *substitutes-varE*:

**assumes** *substitutes (Var x) y M X*

**shows**  $(x = y \wedge M = X) \vee (x \neq y \wedge X = \text{Var } x)$

**by**(

*rule* *substitutes-varE'*[**where**  $x=x$  **and**  $y=y$  **and**  $M=M$  **and**  $X=X$ ],

*metis* *assms*,

*metis unit-not-var,*  
*metis trm-simp(1),*  
*metis trm-simp(1),*  
*auto simp add: var-not-app var-not-fn var-not-pair var-not-fst var-not-snd*  
 )

**inductive-cases** *substitutes-appE'*: *substitutes (App A B) x M X*

**lemma** *substitutes-appE*:

**assumes** *substitutes (App A B) x M X*

**shows**  $\exists A' B'. \text{substitutes } A \ x \ M \ A' \wedge \text{substitutes } B \ x \ M \ B' \wedge X = \text{App } A' \ B'$

**by**(

*cases rule: substitutes-appE'*[**where**  $A=A$  **and**  $B=B$  **and**  $x=x$  **and**  $M=M$  **and**  $X=X$ ],

*metis assms,*

*metis unit-not-app,*

*metis var-not-app,*

*metis var-not-app,*

*metis trm-simp(2,3),*

*auto simp add: app-not-fn app-not-pair app-not-fst app-not-snd*

)

**inductive-cases** *substitutes-fnE'*: *substitutes (Fn y T A) x M X*

**lemma** *substitutes-fnE*:

**assumes** *substitutes (Fn y T A) x M X*  $y \neq x$   $y \notin \text{fvs } M$

**shows**  $\exists A'. \text{substitutes } A \ x \ M \ A' \wedge X = \text{Fn } y \ T \ A'$

**using** *assms* **proof**(*induction rule: substitutes-fnE'*[**where**  $y=y$  **and**  $T=T$  **and**  $A=A$  **and**  $x=x$  **and**  $M=M$  **and**  $X=X$ ])

**case** ( $6 \ z \ B \ B' \ S$ )

**consider**  $y = z \wedge T = S \wedge A = B \mid y \neq z \wedge T = S \wedge y \notin \text{fvs } B \wedge A = [y \leftrightarrow z] \cdot B$

**using**  $\langle \text{Fn } y \ T \ A = \text{Fn } z \ S \ B \rangle$  *trm-simp(4)* **by** *metis*

**thus** *?case* **proof**(*cases*)

**case** 1

**thus** *?thesis* **using** 6 **by** *metis*

**next**

**case** 2

**hence**  $y \neq z \ T = S \ y \notin \text{fvs } B \ A = [y \leftrightarrow z] \cdot B$  **by** *auto*

**have** *substitutes*  $([y \leftrightarrow z] \cdot B) ([y \leftrightarrow z] \ \$ \ x) ([y \leftrightarrow z] \cdot M) ([y \leftrightarrow z] \cdot B')$

**using** *substitutes-prm*  $\langle \text{substitutes } B \ x \ M \ B' \rangle$  **by** *metis*

**hence** *substitutes*  $A ([y \leftrightarrow z] \ \$ \ x) ([y \leftrightarrow z] \cdot M) ([y \leftrightarrow z] \cdot B')$

**using**  $\langle A = [y \leftrightarrow z] \cdot B \rangle$  **by** *metis*

**hence** *substitutes*  $A \ x \ ([y \leftrightarrow z] \cdot M) ([y \leftrightarrow z] \cdot B')$

**using**  $\langle y \neq x \ \langle x \neq z \rangle$  *prm-unit-inaction* **by** *metis*

**hence** \*: *substitutes*  $A \ x \ M \ ([y \leftrightarrow z] \cdot B')$

**using**  $\langle y \notin \text{fvs } M \ \langle z \notin \text{fvs } M \rangle$  *trm-prm-unit-inaction* **by** *metis*

**have**  $y \notin \text{fvs } B'$

**using**

*substitutes-fvs*  $\langle \text{substitutes } B \ x \ M \ B' \ \langle y \notin \text{fvs } B \ \langle y \notin \text{fvs } M \rangle$

```

      Diff-subset UnE rev-subsetD
    by blast
  hence  $X = Fn\ y\ T\ ([y \leftrightarrow z] \cdot B')$ 
    using  $\langle X = Fn\ z\ S\ B' \rangle \langle y \neq z \rangle \langle T = S \rangle$  fn-eq
    by metis

  thus ?thesis using * by auto
next
qed
next
qed (
  metis assms(1),
  metis unit-not-fn,
  metis var-not-fn,
  metis var-not-fn,
  metis app-not-fn,
  metis fn-not-pair,
  metis fn-not-fst,
  metis fn-not-snd
)

inductive-cases substitutes-pairE': substitutes (Pair A B) x M X
lemma substitutes-pairE:
  assumes substitutes (Pair A B) x M X
  shows  $\exists A' B'.\ substitutes\ A\ x\ M\ A' \wedge substitutes\ B\ x\ M\ B' \wedge X = Pair\ A'\ B'$ 
proof(cases rule: substitutes-pairE'[where A=A and B=B and x=x and M=M
and X=X])
  case (7 A A' B B')
    thus ?thesis using trm-simp(5) trm-simp(6) by blast
  next
qed (
  metis assms,
  metis unit-not-pair,
  metis var-not-pair,
  metis var-not-pair,
  metis app-not-pair,
  metis fn-not-pair,
  metis pair-not-fst,
  metis pair-not-snd
)

inductive-cases substitutes-fstE': substitutes (Fst P) x M X
lemma substitutes-fstE:
  assumes substitutes (Fst P) x M X
  shows  $\exists P'.\ substitutes\ P\ x\ M\ P' \wedge X = Fst\ P'$ 
proof(cases rule: substitutes-fstE'[where P=P and x=x and M=M and X=X])
  case (8 P P')
    thus ?thesis using trm-simp(7) by blast
  next

```

```

qed (
  metis assms,
  metis unit-not-fst,
  metis var-not-fst,
  metis var-not-fst,
  metis app-not-fst,
  metis fn-not-fst,
  metis pair-not-fst,
  metis fst-not-snd
)

inductive-cases substitutes-sndE': substitutes (Snd P) x M X
lemma substitutes-sndE:
  assumes substitutes (Snd P) x M X
  shows  $\exists P'. \text{substitutes } P \ x \ M \ P' \wedge X = \text{Snd } P'$ 
proof(cases rule: substitutes-sndE'[where P=P and x=x and M=M and X=X])
  case (9 P P')
    thus ?thesis using trm-simp(8) by blast
  next
qed (
  metis assms,
  metis unit-not-snd,
  metis var-not-snd,
  metis var-not-snd,
  metis app-not-snd,
  metis fn-not-snd,
  metis pair-not-snd,
  metis fst-not-snd
)

lemma substitutes-total:
  shows  $\exists X. \text{substitutes } A \ x \ M \ X$ 
proof(induction A rule: trm-strong-induct[where S={x}  $\cup$  fvs M])
  show finite ({x}  $\cup$  fvs M) using fvs-finite by auto
  next

  case 1
    obtain X :: 'a trm where X = Unit by auto
    thus ?case using substitutes.unit by metis
  next
  case (2 y)
    consider x = y | x  $\neq$  y by auto
    thus ?case proof(cases)
      case 1
        obtain X where X = M by auto
        hence substitutes (Var y) x M X using (x = y) substitutes.var1 by metis
        thus ?thesis by auto
      next
    case 2

```



```

    obtain X where X = (Var y) by auto
    hence substitutes (Var y) x M X using ⟨x ≠ y⟩ substitutes.var2 by metis
    thus ?thesis by auto
  next
  qed
next
case (3 A B)
  from this obtain A' B' where A': substitutes A x M A' and B': substitutes B
x M B' by auto
  obtain X where X = App A' B' by auto
  hence substitutes (App A B) x M X using A' B' substitutes.app by metis
  thus ?case by auto
next
case (4 y T A)
  from this obtain A' where A': substitutes A x M A' by auto
  from ⟨y ∉ ({x} ∪ fvs M)⟩ have y ≠ x y ∉ fvs M by auto
  obtain X where X = Fn y T A' by auto
  hence substitutes (Fn y T A) x M X using substitutes.fn ⟨y ≠ x⟩ ⟨y ∉ fvs M⟩
A' by metis
  thus ?case by auto
next
case (5 A B)
  from this obtain A' B' where substitutes A x M A' substitutes B x M B' by
auto
  from this obtain X where X = Pair A' B' by auto
  hence substitutes (Pair A B) x M X
  using substitutes.pair ⟨substitutes A x M A'⟩ ⟨substitutes B x M B'⟩
  by metis
  thus ?case by auto
next
case (6 P)
  from this obtain P' where substitutes P x M P' by auto
  from this obtain X where X = Fst P' by auto
  hence substitutes (Fst P) x M X using substitutes.fst ⟨substitutes P x M P'⟩
by metis
  thus ?case by auto
next
case (7 P)
  from this obtain P' where substitutes P x M P' by auto
  from this obtain X where X = Snd P' by auto
  hence substitutes (Snd P) x M X using substitutes.snd ⟨substitutes P x M P'⟩
by metis
  thus ?case by auto
next
qed

```

lemma substitutes-unique:

```

  assumes substitutes A x M B substitutes A x M C
  shows B = C

```

```

using assms proof(induction A arbitrary: B C rule: trm-strong-induct[where
S={x} ∪ fvs M])
  show finite ({x} ∪ fvs M) using fvs-finite by auto
  next

  case 1
    thus ?case using substitutes-unitE by metis
  next
  case (2 y)
    thus ?case using substitutes-varE by metis
  next
  case (3 X Y)
    thus ?case using substitutes-appE by metis
  next
  case (4 y T A)
    hence y ≠ x and y ∉ fvs M by auto
    thus ?case using 4 substitutes-fnE by metis
  next
  case (5 A B C D)
    thus ?case using substitutes-pairE by metis
  next
  case (6 P Q R)
    thus ?case using substitutes-fstE by metis
  next
  case (7 P Q R)
    thus ?case using substitutes-sndE by metis
  next
qed

```

```

lemma substitutes-function:
  shows  $\exists! X. \text{substitutes } A \ x \ M \ X$ 
using substitutes-total substitutes-unique by metis

```

```

definition subst :: 'a trm  $\Rightarrow$  'a  $\Rightarrow$  'a trm  $\Rightarrow$  'a trm ([- ::= -]) where
  subst A x M  $\equiv$  (THE X. substitutes A x M X)

```

```

lemma subst-simp-unit:
  shows  $\text{Unit}[x ::= M] = \text{Unit}$ 
unfolding subst-def by(rule, metis substitutes.unit, metis substitutes-function substitutes.unit)

```

```

lemma subst-simp-var1:
  shows  $(\text{Var } x)[x ::= M] = M$ 
unfolding subst-def by(rule, metis substitutes.var1, metis substitutes-function substitutes.var1)

```

```

lemma subst-simp-var2:
  assumes  $x \neq y$ 
  shows  $(\text{Var } x)[y ::= M] = \text{Var } x$ 

```

```

unfolding subst-def by(
  rule,
  metis substitutes.var2 assms,
  metis substitutes-function substitutes.var2 assms
)

lemma subst-simp-app:
  shows  $(App\ A\ B)[x ::= M] = App\ (A[x ::= M])\ (B[x ::= M])$ 
unfolding subst-def proof
  obtain  $A'\ B'$  where  $A': A' = (A[x ::= M])$  and  $B': B' = (B[x ::= M])$  by auto
  hence substitutes A x M A' substitutes B x M B'
    unfolding subst-def
    using substitutes-function theI by metis+
  hence substitutes (App A B) x M (App A' B') using substitutes.app by metis
  thus  $*$ : substitutes (App A B) x M (App (THE X. substitutes A x M X) (THE X. substitutes B x M X))
    using  $A'\ B'$  unfolding subst-def by metis

  fix  $X$ 
  assume substitutes (App A B) x M X
  thus  $X = App\ (THE\ X.\ substitutes\ A\ x\ M\ X)\ (THE\ X.\ substitutes\ B\ x\ M\ X)$ 
    using substitutes-function * by metis
qed

lemma subst-simp-fn:
  assumes  $x \neq y \ y \notin fvs\ M$ 
  shows  $(Fn\ y\ T\ A)[x ::= M] = Fn\ y\ T\ (A[x ::= M])$ 
unfolding subst-def proof
  obtain  $A'$  where  $A': A' = (A[x ::= M])$  by auto
  hence substitutes A x M A' unfolding subst-def using substitutes-function theI
by metis
  hence substitutes (Fn y T A) x M (Fn y T A') using substitutes.fn assms by
metis
  thus  $*$ : substitutes (Fn y T A) x M (Fn y T (THE X. substitutes A x M X))
    using  $A'$  unfolding subst-def by metis

  fix  $X$ 
  assume substitutes (Fn y T A) x M X
  thus  $X = Fn\ y\ T\ (THE\ X.\ substitutes\ A\ x\ M\ X)$  using substitutes-function *
by metis
qed

lemma subst-simp-pair:
  shows  $(Pair\ A\ B)[x ::= M] = Pair\ (A[x ::= M])\ (B[x ::= M])$ 
unfolding subst-def proof
  obtain  $A'\ B'$  where  $A': A' = (A[x ::= M])$  and  $B': B' = (B[x ::= M])$  by auto
  hence substitutes A x M A' substitutes B x M B'
    unfolding subst-def using substitutes-function theI by metis+
  hence substitutes (Pair A B) x M (Pair A' B') using substitutes.pair by metis

```

**thus** \*: *substitutes* (Pair A B) x M (Pair (THE X. *substitutes* A x M X) (THE X. *substitutes* B x M X))

**using** A' B' **unfolding** *subst-def* **by** *metis*

**fix** X

**assume** *substitutes* (Pair A B) x M X

**thus** X = Pair (THE X. *substitutes* A x M X) (THE X. *substitutes* B x M X)

**using** *substitutes-function* \* **by** *metis*

**qed**

**lemma** *subst-simp-fst*:

**shows** (Fst P)[x ::= M] = Fst (P[x ::= M])

**unfolding** *subst-def* **proof**

**obtain** P' **where** P': P' = (P[x ::= M]) **by** *auto*

**hence** *substitutes* P x M P' **unfolding** *subst-def* **using** *substitutes-function theI* **by** *metis*

**hence** *substitutes* (Fst P) x M (Fst P') **using** *substitutes.fst* **by** *metis*

**thus** \*: *substitutes* (Fst P) x M (Fst (THE X. *substitutes* P x M X))

**using** P' **unfolding** *subst-def* **by** *metis*

**fix** X

**assume** *substitutes* (Fst P) x M X

**thus** X = Fst (THE X. *substitutes* P x M X) **using** *substitutes-function* \* **by** *metis*

**qed**

**lemma** *subst-simp-snd*:

**shows** (Snd P)[x ::= M] = Snd (P[x ::= M])

**unfolding** *subst-def* **proof**

**obtain** P' **where** P': P' = (P[x ::= M]) **by** *auto*

**hence** *substitutes* P x M P' **unfolding** *subst-def* **using** *substitutes-function theI* **by** *metis*

**hence** *substitutes* (Snd P) x M (Snd P') **using** *substitutes.snd* **by** *metis*

**thus** \*: *substitutes* (Snd P) x M (Snd (THE X. *substitutes* P x M X))

**using** P' **unfolding** *subst-def* **by** *metis*

**fix** X

**assume** *substitutes* (Snd P) x M X

**thus** X = Snd (THE X. *substitutes* P x M X) **using** *substitutes-function* \* **by** *metis*

**qed**

**lemma** *subst-prm*:

**shows** ( $\pi \cdot (M[z ::= N])$ ) = ( $(\pi \cdot M)[\pi \$ z ::= \pi \cdot N]$ )

**unfolding** *subst-def* **using** *substitutes-prm* *substitutes-function the1-equality* **by** (*metis* (*full-types*))

**lemma** *subst-fvs*:

**shows** *fvs* (M[z ::= N])  $\subseteq$  (*fvs* M - {z})  $\cup$  *fvs* N

**unfolding** *subst-def* **using** *substitutes-fvs substitutes-function theI2* **by** *metis*

**lemma** *subst-free*:

**assumes**  $y \notin \text{fvs } M$

**shows**  $M[y ::= N] = M$

**using** *assms* **proof**(*induction M rule: trm-strong-induct*[**where**  $S = \{y\} \cup \text{fvs } N$ ])

**show** *finite* ( $\{y\} \cup \text{fvs } N$ ) **using** *fvs-finite* **by** *auto*

**case** 1

**thus** *?case* **using** *subst-simp-unit* **by** *metis*

**next**

**case** (2  $x$ )

**thus** *?case* **using** *subst-simp-var2* **by** (*simp add: fvs-simp*)

**next**

**case** (3  $A B$ )

**thus** *?case* **using** *subst-simp-app* **by** (*simp add: fvs-simp*)

**next**

**case** (4  $x T A$ )

**hence**  $y \neq x$   $x \notin \text{fvs } N$  **by** *auto*

**have**  $y \notin \text{fvs } A - \{x\}$  **using**  $\langle y \neq x \rangle \langle y \notin \text{fvs } (Fn\ x\ T\ A) \rangle$  *fvs-simp(4)* **by** *metis*

**hence**  $y \notin \text{fvs } A$  **using**  $\langle y \neq x \rangle$  **by** *auto*

**hence**  $A[y ::= N] = A$  **using** 4.*IH* **by** *blast*

**thus** *?case* **using**  $\langle y \neq x \rangle \langle y \notin \text{fvs } A \rangle \langle x \notin \text{fvs } N \rangle$  *subst-simp-fn* **by** *metis*

**next**

**case** (5  $A B$ )

**thus** *?case* **using** *subst-simp-pair* **by** (*simp add: fvs-simp*)

**next**

**case** (6  $P$ )

**thus** *?case* **using** *subst-simp-fst* **by** (*simp add: fvs-simp*)

**next**

**case** (7  $P$ )

**thus** *?case* **using** *subst-simp-snd* **by** (*simp add: fvs-simp*)

**next**

**qed**

**lemma** *subst-swp*:

**assumes**  $y \notin \text{fvs } A$

**shows**  $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M])$

**using** *assms* **proof**(*induction A rule: trm-strong-induct*[**where**  $S = \{y, z\} \cup \text{fvs } M$ ])

**show** *finite* ( $\{y, z\} \cup \text{fvs } M$ ) **using** *fvs-finite* **by** *auto*

**next**

**case** 1

**thus** *?case* **using** *trm-prm-simp(1)* *subst-simp-unit* **by** *metis*

**next**

**case** (2  $x$ )

```

hence  $y \neq x$  using fvs-simp(2) by force
consider  $x = z \mid x \neq z$  by auto
thus ?case proof(cases)
  case 1
    thus ?thesis using subst-simp-var1 trm-prm-simp(2) prm-unit-action
prm-unit-commutes by metis
  next
  case 2
    thus ?thesis using subst-simp-var2 trm-prm-simp(2) prm-unit-inaction  $\langle y \neq x \rangle$  by metis
  next
  qed
next
case ( $\exists A B$ )
  from  $\langle y \notin \text{fvs } (App A B) \rangle$  have  $y \notin \text{fvs } A \ y \notin \text{fvs } B$  by (auto simp add: fvs-simp(3))
  thus ?case using 3.IH subst-simp-app trm-prm-simp(3) by metis
next
case ( $\lambda x T A$ )
  hence  $x \neq y \ x \neq z \ x \notin \text{fvs } M$  by auto
  hence  $y \notin \text{fvs } A$  using  $\langle y \notin \text{fvs } (Fn x T A) \rangle$  fvs-simp(4) by force
  hence  $*$ :  $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M])$  using 4.IH by metis

  have  $([y \leftrightarrow z] \cdot Fn x T A)[y ::= M] = ((Fn ([y \leftrightarrow z] \$ x) T ([y \leftrightarrow z] \cdot A))[y ::= M])$ 
  using trm-prm-simp(4) by metis
  also have  $\dots = ((Fn x T ([y \leftrightarrow z] \cdot A))[y ::= M])$ 
  using prm-unit-inaction  $\langle x \neq y \rangle \langle x \neq z \rangle$  by metis
  also have  $\dots = Fn x T ([y \leftrightarrow z] \cdot A)[y ::= M]$ 
  using subst-simp-fn  $\langle x \neq y \rangle \langle x \notin \text{fvs } M \rangle$  by metis
  also have  $\dots = Fn x T (A[z ::= M])$  using  $*$  by metis
  also have  $\dots = ((Fn x T A)[z ::= M])$ 
  using subst-simp-fn  $\langle x \neq z \rangle \langle x \notin \text{fvs } M \rangle$  by metis
  finally show ?case.
next
case ( $\exists A B$ )
  from  $\langle y \notin \text{fvs } (Pair A B) \rangle$  have  $y \notin \text{fvs } A \ y \notin \text{fvs } B$  by (auto simp add: fvs-simp(5))
  hence  $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M]) \ ([y \leftrightarrow z] \cdot B)[y ::= M] = (B[z ::= M])$ 
  using 5.IH by metis+
  thus ?case using trm-prm-simp(5) subst-simp-pair by metis
next
case ( $\exists P$ )
  from  $\langle y \notin \text{fvs } (Fst P) \rangle$  have  $y \notin \text{fvs } P$  by (simp add: fvs-simp(6))
  hence  $([y \leftrightarrow z] \cdot P)[y ::= M] = (P[z ::= M])$  using 6.IH by metis
  thus ?case using trm-prm-simp(6) subst-simp-fst by metis
next
case ( $\exists P$ )

```

```

    from  $\langle y \notin fvs (Snd P) \rangle$  have  $y \notin fvs P$  by (simp add: fvs-simp(7))
    hence  $([y \leftrightarrow z] \cdot P)[y ::= M] = (P[z ::= M])$  using 7.IH by metis
    thus ?case using trm-prm-simp(7) subst-simp-snd by metis
  next
qed

lemma barendregt:
  assumes  $y \neq z \ y \notin fvs L$ 
  shows  $M[y ::= N][z ::= L] = (M[z ::= L][y ::= N[z ::= L]])$ 
using assms proof(induction M rule: trm-strong-induct[where  $S = \{y, z\} \cup fvs N \cup fvs L$ ])
  show finite  $(\{y, z\} \cup fvs N \cup fvs L)$  using fvs-finite by auto
  next

  case 1
    thus ?case using subst-simp-unit by metis
  next
  case (2 x)
    consider  $x = y \mid x = z \mid x \neq y \wedge x \neq z$  by auto
    thus ?case proof(cases)
      case 1
        hence  $x = y \ x \neq z$  using assms(1) by auto
        thus ?thesis using subst-simp-var1 subst-simp-var2 by metis
      next
      case 2
        hence  $x \neq y \ x = z$  using assms(1) by auto
        thus ?thesis using subst-free  $\langle y \notin fvs L \rangle$  subst-simp-var1 subst-simp-var2
    by metis
    next
    case 3
      then show ?thesis using subst-simp-var2 by metis
    next
  qed
  next
  case (3 A B)
    thus ?case using subst-simp-app by metis
  next
  case (4 x T A)
    hence  $*$ :  $A[y ::= N][z ::= L] = (A[z ::= L][y ::= N[z ::= L]])$  by blast
    from  $\langle x \notin \{y, z\} \cup fvs N \cup fvs L \rangle$  have  $x \neq y \ x \neq z \ x \notin fvs N \ x \notin fvs L$  by
  auto
    hence  $x \notin fvs (N[z ::= L])$  using subst-fvs by auto

  have  $(Fn\ x\ T\ A)[y ::= N][z ::= L] = Fn\ x\ T\ (A[y ::= N][z ::= L])$ 
    using subst-simp-fn  $\langle x \neq y \rangle \langle x \neq z \rangle \langle x \notin fvs N \rangle \langle x \notin fvs L \rangle$  by metis
  also have  $\dots = Fn\ x\ T\ (A[z ::= L][y ::= N[z ::= L]])$  using  $*$  by metis
  also have  $\dots = ((Fn\ x\ T\ A)[z ::= L][y ::= N[z ::= L]])$ 
    using subst-simp-fn  $\langle x \neq y \rangle \langle x \neq z \rangle \langle x \notin fvs (N[z ::= L]) \rangle \langle x \notin fvs L \rangle$  by
  metis

```

```

    finally show ?case.
  next
  case (5 A B)
    thus ?case using subst-simp-pair by metis
  next
  case (6 P)
    thus ?case using subst-simp-fst by metis
  next
  case (7 P)
    thus ?case using subst-simp-snd by metis
  next
qed

lemma typing-subst:
  assumes  $\Gamma(z \mapsto \tau) \vdash M : \sigma$   $\Gamma \vdash N : \tau$ 
  shows  $\Gamma \vdash M[z ::= N] : \sigma$ 
using assms proof(induction M arbitrary:  $\Gamma \sigma$  rule: trm-strong-depth-induct[where
 $S = \{z\} \cup fvs N$ ])
  show finite ( $\{z\} \cup fvs N$ ) using fvs-finite by auto
  next

  case 1
    thus ?case using subst-simp-unit typing.tunit typing-unitE by metis
  next
  case (2 x)
    hence *:  $(\Gamma(z \mapsto \tau)) x = \text{Some } \sigma$  using typing-varE by metis

    consider  $x = z \mid x \neq z$  by auto
    thus ?case proof(cases)
      case 1
        hence  $(\Gamma(x \mapsto \tau)) x = \text{Some } \sigma$  using * by metis
        hence  $\tau = \sigma$  by auto
        thus ?thesis using  $\langle \Gamma \vdash N : \tau \rangle$  subst-simp-var1  $\langle x = z \rangle$  by metis
      next
      case 2
        hence  $\Gamma x = \text{Some } \sigma$  using * by auto
        hence  $\Gamma \vdash \text{Var } x : \sigma$  using typing.tvar by metis
        thus ?thesis using  $\langle x \neq z \rangle$  subst-simp-var2 by metis
    next
  qed
  next
  case (3 A B)
    have *:  $\text{depth } A < \text{depth } (\text{App } A B) \wedge \text{depth } B < \text{depth } (\text{App } A B)$ 
      using depth-app by auto

    from  $\langle \Gamma(z \mapsto \tau) \vdash \text{App } A B : \sigma \rangle$  obtain  $\sigma'$  where
       $\Gamma(z \mapsto \tau) \vdash A : (\text{TArr } \sigma' \sigma)$ 
       $\Gamma(z \mapsto \tau) \vdash B : \sigma'$ 
      using typing-appE by metis

```



**hence**  
 $\Gamma \vdash (A[z ::= N]) : (TArr \sigma' \sigma)$   
 $\Gamma \vdash (B[z ::= N]) : \sigma'$   
**using**  $\mathcal{B} *$  **by** *metis+*  
**hence**  $\Gamma \vdash App (A[z ::= N]) (B[z ::= N]) : \sigma$  **using** *typing.tapp* **by** *metis*  
**thus** *?case* **using** *subst-simp-app* **by** *metis*  
**next**  
**case** ( $\mathcal{4} x T A$ )  
**hence**  $x \neq z$   $x \notin fvs N$  **by** *auto*  
**hence**  $*$ :  $\Gamma(x \mapsto T) \vdash N : \tau$  **using** *typing-weaken*  $\mathcal{4}$  **by** *metis*  
**have**  $**$ :  $depth A < depth (Fn x T A)$  **using** *depth-fn*.  
  
**from**  $\langle \Gamma(z \mapsto \tau) \vdash Fn x T A : \sigma \rangle$  **obtain**  $\sigma'$  **where**  
 $\sigma = TArr T \sigma'$   
 $\Gamma(z \mapsto \tau)(x \mapsto T) \vdash A : \sigma'$   
**using** *typing-fnE* **by** *metis*  
**hence**  $\Gamma(x \mapsto T)(z \mapsto \tau) \vdash A : \sigma'$  **using**  $\langle x \neq z \rangle$  *fun-upd-twist* **by** *metis*  
**hence**  $\Gamma(x \mapsto T) \vdash A[z ::= N] : \sigma'$  **using**  $\mathcal{4} * **$  **by** *metis*  
**hence**  $\Gamma \vdash Fn x T (A[z ::= N]) : \sigma$  **using** *typing.tfn*  $\langle \sigma = TArr T \sigma' \rangle$  **by** *metis*  
**thus** *?case* **using**  $\langle x \neq z \rangle \langle x \notin fvs N \rangle$  *subst-simp-fn* **by** *metis*  
**next**  
**case** ( $\mathcal{5} A B$ )  
**from** *this* **obtain**  $T S$  **where**  $\sigma = TPair T S$   $\Gamma(z \mapsto \tau) \vdash A : T$   $\Gamma(z \mapsto \tau) \vdash B : S$   
**using** *typing-pairE* **by** *metis*  
**moreover** **have**  $depth A < depth (Pair A B)$  **and**  $depth B < depth (Pair A B)$   
**using** *depth-pair* **by** *auto*  
**ultimately** **have**  $\Gamma \vdash A[z ::= N] : T$   $\Gamma \vdash B[z ::= N] : S$  **using**  $\mathcal{5}$  **by** *metis+*  
**hence**  $\Gamma \vdash Pair (A[z ::= N]) (B[z ::= N]) : \sigma$  **using**  $\langle \sigma = TPair T S \rangle$  *typing.tpair* **by** *metis*  
**thus** *?case* **using** *subst-simp-pair* **by** *metis*  
**next**  
**case** ( $\mathcal{6} P$ )  
**from** *this* **obtain**  $\sigma'$  **where**  $\Gamma(z \mapsto \tau) \vdash P : (TPair \sigma \sigma')$  **using** *typing-fstE* **by** *metis*  
**moreover** **have**  $depth P < depth (Fst P)$  **using** *depth-fst* **by** *metis*  
**ultimately** **have**  $\Gamma \vdash P[z ::= N] : (TPair \sigma \sigma')$  **using**  $\mathcal{6}$  **by** *metis*  
**hence**  $\Gamma \vdash Fst (P[z ::= N]) : \sigma$  **using** *typing.tfst* **by** *metis*  
**thus** *?case* **using** *subst-simp-fst* **by** *metis*  
**next**  
**case** ( $\mathcal{7} P$ )  
**from** *this* **obtain**  $\sigma'$  **where**  $\Gamma(z \mapsto \tau) \vdash P : (TPair \sigma' \sigma)$  **using** *typing-sndE* **by** *metis*  
**moreover** **have**  $depth P < depth (Snd P)$  **using** *depth-snd* **by** *metis*  
**ultimately** **have**  $\Gamma \vdash P[z ::= N] : (TPair \sigma' \sigma)$  **using**  $\mathcal{7}$  **by** *metis*  
**hence**  $\Gamma \vdash Snd (P[z ::= N]) : \sigma$  **using** *typing.tsnd* **by** *metis*  
**thus** *?case* **using** *subst-simp-snd* **by** *metis*  
**next**

qed

**inductive** *beta-reduction* :: 'a trm  $\Rightarrow$  'a trm  $\Rightarrow$  bool ( $- \rightarrow\beta -$ ) **where**

*beta*: (App (Fn x T A) M)  $\rightarrow\beta$  (A[x ::= M])  
| *app1*: A  $\rightarrow\beta$  A'  $\Longrightarrow$  (App A B)  $\rightarrow\beta$  (App A' B)  
| *app2*: B  $\rightarrow\beta$  B'  $\Longrightarrow$  (App A B)  $\rightarrow\beta$  (App A B')  
| *fn*: A  $\rightarrow\beta$  A'  $\Longrightarrow$  (Fn x T A)  $\rightarrow\beta$  (Fn x T A')  
| *pair1*: A  $\rightarrow\beta$  A'  $\Longrightarrow$  (Pair A B)  $\rightarrow\beta$  (Pair A' B)  
| *pair2*: B  $\rightarrow\beta$  B'  $\Longrightarrow$  (Pair A B)  $\rightarrow\beta$  (Pair A B')  
| *fst1*: P  $\rightarrow\beta$  P'  $\Longrightarrow$  (Fst P)  $\rightarrow\beta$  (Fst P')  
| *fst2*: (Fst (Pair A B))  $\rightarrow\beta$  A  
| *snd1*: P  $\rightarrow\beta$  P'  $\Longrightarrow$  (Snd P)  $\rightarrow\beta$  (Snd P')  
| *snd2*: (Snd (Pair A B))  $\rightarrow\beta$  B

**lemma** *beta-reduction-fvs*:

**assumes** M  $\rightarrow\beta$  M'  
**shows** fvs M'  $\subseteq$  fvs M  
**using** *assms* **proof** (*induction*)  
**case** (*beta* x T A M)  
  **thus** ?*case* **using** *fvs-simp*(3) *fvs-simp*(4) *subst-fvs* **by** *metis*  
**next**  
**case** (*app1* A A' B)  
  **hence** fvs A'  $\cup$  fvs B  $\subseteq$  fvs A  $\cup$  fvs B **by** *auto*  
  **thus** ?*case* **using** *fvs-simp*(3) **by** *metis*  
**next**  
**case** (*app2* B B' A)  
  **hence** fvs A  $\cup$  fvs B'  $\subseteq$  fvs A  $\cup$  fvs B **by** *auto*  
  **thus** ?*case* **using** *fvs-simp*(3) **by** *metis*  
**next**  
**case** (*fn* A A' x T)  
  **hence** fvs A' - {x}  $\subseteq$  fvs A - {x} **by** *auto*  
  **thus** ?*case* **using** *fvs-simp*(4) **by** *metis*  
**next**  
**case** (*pair1* A A' B)  
  **hence** fvs A'  $\cup$  fvs B  $\subseteq$  fvs A  $\cup$  fvs B **by** *auto*  
  **thus** ?*case* **using** *fvs-simp*(5) **by** *metis*  
**next**  
**case** (*pair2* B B' A)  
  **hence** fvs A  $\cup$  fvs B'  $\subseteq$  fvs A  $\cup$  fvs B **by** *auto*  
  **thus** ?*case* **using** *fvs-simp*(5) **by** *metis*  
**next**  
**case** (*fst1* P P')  
  **thus** ?*case* **using** *fvs-simp*(6) **by** *metis*  
**next**  
**case** (*fst2* A B)  
  **thus** ?*case* **using** *fvs-simp*(5, 6) **by** *force*  
**next**  
**case** (*snd1* P P')

```

    thus ?case using fvs-simp(7) by metis
  next
  case (snd2 A B)
    thus ?case using fvs-simp(5, 7) by force
  next
qed

```

```

lemma beta-reduction-prm:
  assumes  $M \rightarrow\beta M'$ 
  shows  $(\pi \cdot M) \rightarrow\beta (\pi \cdot M')$ 
using assms by(induction, auto simp add: beta-reduction.intros trm-prm-simp
subst-prm)

```

```

lemma beta-reduction-left-unitE:
  assumes  $Unit \rightarrow\beta M'$ 
  shows False
using assms by(cases, auto simp add: unit-not-app unit-not-fn unit-not-pair unit-not-fst
unit-not-snd)

```

```

lemma beta-reduction-left-varE:
  assumes  $(Var\ x) \rightarrow\beta M'$ 
  shows False
using assms by(cases, auto simp add: var-not-app var-not-fn var-not-pair var-not-fst
var-not-snd)

```

```

lemma beta-reduction-left-appE:
  assumes  $(App\ A\ B) \rightarrow\beta M'$ 
  shows
    ( $\exists x\ T\ X. A = (Fn\ x\ T\ X) \wedge M' = (X[x ::= B])$ )  $\vee$ 
    ( $\exists A'. (A \rightarrow\beta A') \wedge M' = App\ A'\ B$ )  $\vee$ 
    ( $\exists B'. (B \rightarrow\beta B') \wedge M' = App\ A\ B'$ )

```

```

using assms by(
  cases,
  metis trm-simp(2, 3),
  metis trm-simp(2, 3),
  metis trm-simp(2, 3),
  auto simp add: app-not-fn app-not-pair app-not-fst app-not-snd
)

```

```

lemma beta-reduction-left-fnE:
  assumes  $(Fn\ x\ T\ A) \rightarrow\beta M'$ 
  shows  $\exists A'. (A \rightarrow\beta A') \wedge M' = (Fn\ x\ T\ A')$ 
using assms proof(cases)
  case (fn B B' y S)
    consider  $x = y \wedge T = S \wedge A = B \mid x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x$ 
 $\leftrightarrow y] \cdot B$ 
    using trm-simp(4) ( $Fn\ x\ T\ A = Fn\ y\ S\ B$ ) by metis
    thus ?thesis proof(cases)

```

```

    case 1
      thus ?thesis using fn by auto
    next
    case 2
      thus ?thesis using fn beta-reduction-fvs beta-reduction-prm fn-eq by fastforce
    next
  qed
next
qed (
  metis app-not-fn,
  metis app-not-fn,
  metis app-not-fn,
  auto simp add: fn-not-pair fn-not-fst fn-not-snd
)

lemma beta-reduction-left-pairE:
  assumes (Pair A B)  $\rightarrow\beta$  M'
  shows  $(\exists A'. (A \rightarrow\beta A') \wedge M' = (\text{Pair } A' B)) \vee (\exists B'. (B \rightarrow\beta B') \wedge M' = (\text{Pair } A B'))$ 
using assms
  apply cases
  prefer 5
  apply (metis trm-simp(5, 6))
  prefer 5
  apply (metis trm-simp(5, 6))
  apply (metis app-not-pair, metis app-not-pair, metis app-not-pair, metis fn-not-pair, metis pair-not-fst, metis pair-not-fst, metis pair-not-snd, metis pair-not-snd)
done

lemma beta-reduction-left-fstE:
  assumes (Fst P)  $\rightarrow\beta$  M'
  shows  $(\exists P'. (P \rightarrow\beta P') \wedge M' = (\text{Fst } P')) \vee (\exists A B. P = (\text{Pair } A B) \wedge M' = A)$ 
using assms proof(cases)
  case (fst1 P P')
    thus ?thesis using trm-simp(7) by blast
  next
  case (fst2 B)
    thus ?thesis using trm-simp(7) by blast
  next
qed (
  metis app-not-fst,
  metis app-not-fst,
  metis app-not-fst,
  metis fn-not-fst,
  metis pair-not-fst,
  metis pair-not-fst,
  metis fst-not-snd,
  metis fst-not-snd
)

```

)

**lemma** *beta-reduction-left-sndE*:

**assumes**  $(Snd\ P) \rightarrow\beta\ M'$

**shows**  $(\exists P'. (P \rightarrow\beta\ P') \wedge M' = (Snd\ P')) \vee (\exists A\ B. P = Pair\ A\ B \wedge M' = B)$

**using** *assms* **proof**(*cases*)

**case** (*snd1*  $P\ P'$ )

**thus** *?thesis* **using** *trm-simp*(8) **by** *blast*

**next**

**case** (*snd2*  $A$ )

**thus** *?thesis* **using** *trm-simp*(8) **by** *blast*

**next**

**qed** (

*metis* *app-not-snd*,

*metis* *app-not-snd*,

*metis* *app-not-snd*,

*metis* *fn-not-snd*,

*metis* *pair-not-snd*,

*metis* *pair-not-snd*,

*metis* *fst-not-snd*,

*metis* *fst-not-snd*

)

**lemma** *preservation'*:

**assumes**  $\Gamma \vdash M : \tau$  **and**  $M \rightarrow\beta\ M'$

**shows**  $\Gamma \vdash M' : \tau$

**using** *assms* **proof**(*induction arbitrary: M' rule: typing.induct*)

**case** (*tunit*  $\Gamma$ )

**thus** *?case* **using** *beta-reduction-left-unitE* **by** *metis*

**next**

**case** (*tvar*  $\Gamma\ x\ \tau$ )

**thus** *?case* **using** *beta-reduction-left-varE* **by** *metis*

**next**

**case** (*tapp*  $\Gamma\ A\ \tau\ \sigma\ B\ M'$ )

**from**  $\langle (App\ A\ B) \rightarrow\beta\ M' \rangle$  **consider**

$(\exists x\ T\ X. A = (Fn\ x\ T\ X) \wedge M' = (X[x ::= B])) \mid$

$(\exists A'. (A \rightarrow\beta\ A') \wedge M' = App\ A'\ B) \mid$

$(\exists B'. (B \rightarrow\beta\ B') \wedge M' = App\ A\ B')$  **using** *beta-reduction-left-appE* **by** *metis*

**thus** *?case* **proof**(*cases*)

**case** *1*

**from** *this* **obtain**  $x\ T\ X$  **where**  $A = Fn\ x\ T\ X$  **and**  $*$ :  $M' = (X[x ::= B])$

**by** *auto*

**have**  $\Gamma(x \mapsto \tau) \vdash X : \sigma$  **using** *typing-fnE*  $\langle \Gamma \vdash A : (TArr\ \tau\ \sigma) \rangle$   $\langle A = Fn\ x\ T\ X \rangle$  *type.inject*

**by** *blast*

**hence**  $\Gamma \vdash (X[x ::= B]) : \sigma$  **using** *typing-subst*  $\langle \Gamma \vdash B : \tau \rangle$  **by** *metis*

```

    thus ?thesis using * by auto
  next
  case 2
    from this obtain A' where A  $\rightarrow\beta$  A' and *: M' = (App A' B) by auto
    hence  $\Gamma \vdash A' : (TArr \tau \sigma)$  using tapp.IH(1) by metis
    hence  $\Gamma \vdash (App A' B) : \sigma$  using  $\langle \Gamma \vdash B : \tau \rangle$  typing.tapp by metis
    thus ?thesis using * by auto
  next
  case 3
    from this obtain B' where B  $\rightarrow\beta$  B' and *: M' = (App A B') by auto
    hence  $\Gamma \vdash B' : \tau$  using tapp.IH(2) by metis
    hence  $\Gamma \vdash (App A B') : \sigma$  using  $\langle \Gamma \vdash A : (TArr \tau \sigma) \rangle$  typing.tapp by metis
    thus ?thesis using * by auto
  next
  qed
next
case (tfn  $\Gamma x T A \sigma$ )
  from this obtain A' where A  $\rightarrow\beta$  A' and *: M' = (Fn x T A')
  using beta-reduction-left-fnE by metis
  hence  $\Gamma(x \mapsto T) \vdash A' : \sigma$  using tfn.IH by metis
  hence  $\Gamma \vdash (Fn x T A') : (TArr T \sigma)$  using typing.tfn by metis
  thus ?case using * by auto
next
case (tpair  $\Gamma A \tau B \sigma$ )
  from this consider
   $\exists A'. (A \rightarrow\beta A') \wedge M' = (Pair A' B)$ 
  |  $\exists B'. (B \rightarrow\beta B') \wedge M' = (Pair A B')$ 
  using beta-reduction-left-pairE by metis
  thus ?case proof(cases)
  case 1
    from this obtain A' where A  $\rightarrow\beta$  A' and M' = Pair A' B by auto
    thus ?thesis using tpair typing.tpair by metis
  next
  case 2
    from this obtain B' where B  $\rightarrow\beta$  B' and M' = Pair A B' by auto
    thus ?thesis using tpair typing.tpair by metis
  next
  qed
next
case (tfst  $\Gamma P \tau \sigma$ )
  from this consider
   $\exists P'. (P \rightarrow\beta P') \wedge M' = Fst P'$ 
  |  $\exists A B. P = Pair A B \wedge M' = A$  using beta-reduction-left-fstE by metis
  thus ?case proof(cases)
  case 1
    from this obtain P' where P  $\rightarrow\beta$  P' and M' = Fst P' by auto
    thus ?thesis using tfst typing.tfst by metis
  next
  case 2

```

**from this obtain**  $A B$  **where**  $P = \text{Pair } A B$  **and**  $M' = A$  **by** *auto*  
**thus** *?thesis* **using**  $\langle \Gamma \vdash P : (\text{TPair } \tau \sigma) \rangle$  *typing-pairE* **by** *blast*  
**next**  
**qed**  
**next**  
**case**  $(\text{tsnd } \Gamma P \tau \sigma)$   
**from this consider**  
 $\exists P'. (P \rightarrow\beta P') \wedge M' = \text{Snd } P'$   
 $|\exists A B. P = \text{Pair } A B \wedge M' = B$  **using** *beta-reduction-left-sndE* **by** *metis*  
**thus** *?case* **proof**(*cases*)  
**case** 1  
**from this obtain**  $P'$  **where**  $P \rightarrow\beta P'$  **and**  $M' = \text{Snd } P'$  **by** *auto*  
**thus** *?thesis* **using** *tsnd typing.tsnd* **by** *metis*  
**next**  
**case** 2  
**from this obtain**  $A B$  **where**  $P = \text{Pair } A B$  **and**  $M' = B$  **by** *auto*  
**thus** *?thesis* **using**  $\langle \Gamma \vdash P : (\text{TPair } \tau \sigma) \rangle$  *typing-pairE* **by** *blast*  
**next**  
**qed**  
**next**  
**qed**

**inductive**  $NF :: 'a \text{ trm} \Rightarrow \text{bool}$  **where**  
 $\text{unit}: NF \text{ Unit}$   
 $|\text{var}: NF (\text{Var } x)$   
 $|\text{app}: \llbracket A \neq \text{Fn } x T A'; NF A; NF B \rrbracket \Longrightarrow NF (\text{App } A B)$   
 $|\text{fn}: NF A \Longrightarrow NF (\text{Fn } x T A)$   
 $|\text{pair}: \llbracket NF A; NF B \rrbracket \Longrightarrow NF (\text{Pair } A B)$   
 $|\text{fst}: \llbracket P \neq \text{Pair } A B; NF P \rrbracket \Longrightarrow NF (\text{Fst } P)$   
 $|\text{snd}: \llbracket P \neq \text{Pair } A B; NF P \rrbracket \Longrightarrow NF (\text{Snd } P)$

**theorem** *progress*:  
**assumes**  $\Gamma \vdash M : \tau$   
**shows**  $NF M \vee (\exists M'. (M \rightarrow\beta M'))$   
**using** *assms* **proof**(*induction M arbitrary:  $\Gamma \tau$  rule: trm-induct*)  
**case** 1  
**thus** *?case* **using** *NF.unit* **by** *metis*  
**next**  
**case**  $(2 x)$   
**thus** *?case* **using** *NF.var* **by** *metis*  
**next**  
**case**  $(3 A B)$   
**from**  $\langle \Gamma \vdash \text{App } A B : \tau \rangle$  **obtain**  $\sigma$   
**where**  $\Gamma \vdash A : (\text{TArr } \sigma \tau)$  **and**  $\Gamma \vdash B : \sigma$   
**using** *typing-appE* **by** *metis*  
**hence**  $A: NF A \vee (\exists A'. (A \rightarrow\beta A'))$  **and**  $B: NF B \vee (\exists B'. (B \rightarrow\beta B'))$   
**using** 3.IH **by** *auto*

**consider**  $NF B | \exists B'. (B \rightarrow\beta B')$  **using**  $B$  **by** *auto*

```

thus ?case proof(cases)
  case 1
    consider  $NF\ A \mid \exists A'. (A \rightarrow\beta\ A')$  using  $A$  by auto
    thus ?thesis proof(cases)
      case 1
        consider  $\exists x\ T\ A'. A = Fn\ x\ T\ A' \mid \nexists x\ T\ A'. A = Fn\ x\ T\ A'$  by auto
        thus ?thesis proof(cases)
          case 1
            from this obtain  $x\ T\ A'$  where  $A = Fn\ x\ T\ A'$  by auto
            hence  $(App\ A\ B) \rightarrow\beta\ (A'[x ::= B])$  using beta-reduction.beta by
metis
              thus ?thesis by blast
            next
          case 2
            thus ?thesis using  $\langle NF\ A \rangle\ \langle NF\ B \rangle\ NF.app$  by metis
          next
        qed
      next
    case 2
      thus ?thesis using beta-reduction.app1 by metis
    next
  qed
next
case 2
  thus ?thesis using beta-reduction.app2 by metis
next
qed
next
case ( $\lambda x\ T\ A$ )
  from  $\langle \Gamma \vdash Fn\ x\ T\ A : \tau \rangle$  obtain  $\sigma$ 
  where  $\tau = TArr\ T\ \sigma$  and  $\Gamma(x \mapsto T) \vdash A : \sigma$ 
  using typing-fnE by metis
  from  $\langle \Gamma(x \mapsto T) \vdash A : \sigma \rangle$  consider  $NF\ A \mid \exists A'. (A \rightarrow\beta\ A')$ 
  using  $\lambda.IH$  by metis

thus ?case proof(cases)
  case 1
    thus ?thesis using NF.fn by metis
  next
case 2
    from this obtain  $A'$  where  $A \rightarrow\beta\ A'$  by auto
    obtain  $M'$  where  $M' = Fn\ x\ T\ A'$  by auto
    hence  $(Fn\ x\ T\ A) \rightarrow\beta\ M'$  using  $\langle A \rightarrow\beta\ A' \rangle$  beta-reduction.fn by metis
    thus ?thesis by auto
  next
qed
next
case ( $\lambda A\ B$ )
  thus ?case using typing-pairE beta-reduction.pair1 beta-reduction.pair2 NF.pair

```



```

by meson
next
case (6 P)
  from this consider NF P |  $\exists P'. (P \rightarrow\beta P')$  using typing-fstE by metis
  thus ?case proof(cases)
  case 1
    consider  $\exists A B. P = \text{Pair } A B$  |  $\nexists A B. P = \text{Pair } A B$  by auto
    thus ?thesis proof(cases)
    case 1
      from this obtain A B where  $P = \text{Pair } A B$  by auto
      hence  $(\text{Fst } P) \rightarrow\beta A$  using beta-reduction.fst2 by metis
      thus ?thesis by auto
    next
    case 2
      thus ?thesis using  $\langle NF P \rangle NF.fst$  by metis
    next
  qed
next
case 2
  thus ?thesis using beta-reduction.fst1 by metis
next
qed
next
case (7 P)
  from this consider NF P |  $\exists P'. (P \rightarrow\beta P')$  using typing-sndE by metis
  thus ?case proof(cases)
  case 1
    consider  $\exists A B. P = \text{Pair } A B$  |  $\nexists A B. P = \text{Pair } A B$  by auto
    thus ?thesis proof(cases)
    case 1
      from this obtain A B where  $P = \text{Pair } A B$  by auto
      hence  $(\text{Snd } P) \rightarrow\beta B$  using beta-reduction.snd2 by metis
      thus ?thesis by auto
    next
    case 2
      thus ?thesis using  $\langle NF P \rangle NF.snd$  by metis
    next
  qed
next
case 2
  thus ?thesis using beta-reduction.snd1 by metis
next
qed
next
qed

```

**inductive** beta-reduces :: 'a trm  $\Rightarrow$  'a trm  $\Rightarrow$  bool ( $- \rightarrow\beta^* -$ ) **where**  
*reflexive:*  $M \rightarrow\beta^* M$   
*transitive:*  $\llbracket M \rightarrow\beta^* M'; M' \rightarrow\beta M'' \rrbracket \Longrightarrow M \rightarrow\beta^* M''$

```

lemma beta-reduces-composition:
  assumes  $A' \rightarrow\beta^* A''$  and  $A \rightarrow\beta^* A'$ 
  shows  $A \rightarrow\beta^* A''$ 
using assms proof(induction)
  case (reflexive  $B$ )
    thus ?case.
  next
  case (transitive  $B B' B''$ )
    thus ?case using beta-reduces.transitive by metis
  next
qed

lemma beta-reduces-fvs:
  assumes  $A \rightarrow\beta^* A'$ 
  shows  $fvs A' \subseteq fvs A$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case by auto
  next
  case (transitive  $M M' M''$ )
    hence  $fvs M'' \subseteq fvs M'$  using beta-reduction-fvs by metis
    thus ?case using  $\langle fvs M' \subseteq fvs M \rangle$  by auto
  next
qed

lemma beta-reduces-app-left:
  assumes  $A \rightarrow\beta^* A'$ 
  shows  $(App A B) \rightarrow\beta^* (App A' B)$ 
using assms proof(induction)
  case (reflexive  $A$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $A A' A''$ )
    thus ?case using beta-reduces.transitive beta-reduction.app1 by metis
  next
qed

lemma beta-reduces-app-right:
  assumes  $B \rightarrow\beta^* B'$ 
  shows  $(App A B) \rightarrow\beta^* (App A B')$ 
using assms proof(induction)
  case (reflexive  $B$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $B B' B''$ )
    thus ?case using beta-reduces.transitive beta-reduction.app2 by metis
  next
qed

```

```

lemma beta-reduces-fn:
  assumes  $A \rightarrow\beta^* A'$ 
  shows  $(Fn\ x\ T\ A) \rightarrow\beta^* (Fn\ x\ T\ A')$ 
using assms proof(induction)
  case (reflexive  $A$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $A\ A'\ A''$ )
    thus ?case using beta-reduces.transitive beta-reduction.fn by metis
  next
qed

lemma beta-reduces-pair-left:
  assumes  $A \rightarrow\beta^* A'$ 
  shows  $(Pair\ A\ B) \rightarrow\beta^* (Pair\ A'\ B)$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $M\ M'\ M''$ )
    thus ?case using beta-reduces.transitive beta-reduction.pair1 by metis
  next
qed

lemma beta-reduces-pair-right:
  assumes  $B \rightarrow\beta^* B'$ 
  shows  $(Pair\ A\ B) \rightarrow\beta^* (Pair\ A\ B')$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $M\ M'\ M''$ )
    thus ?case using beta-reduces.transitive beta-reduction.pair2 by metis
  next
qed

lemma beta-reduces-fst:
  assumes  $P \rightarrow\beta^* P'$ 
  shows  $(Fst\ P) \rightarrow\beta^* (Fst\ P')$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $M\ M'\ M''$ )
    thus ?case using beta-reduces.transitive beta-reduction.fst1 by metis
  next
qed

```

```

lemma beta-reduces-snd:
  assumes  $P \rightarrow\beta^* P'$ 
  shows  $(Snd P) \rightarrow\beta^* (Snd P')$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case using beta-reduces.reflexive.
  next
  case (transitive  $M M' M''$ )
    thus ?case using beta-reduces.transitive beta-reduction.snd1 by metis
  next
qed

```

```

theorem preservation:
  assumes  $M \rightarrow\beta^* M' \Gamma \vdash M : \tau$ 
  shows  $\Gamma \vdash M' : \tau$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case.
  next
  case (transitive  $M M' M''$ )
    thus ?case using preservation' by metis
  next
qed

```

```

theorem safety:
  assumes  $M \rightarrow\beta^* M' \Gamma \vdash M : \tau$ 
  shows  $NF M' \vee (\exists M''. (M' \rightarrow\beta M''))$ 
using assms proof(induction)
  case (reflexive  $M$ )
    thus ?case using progress by metis
  next
  case (transitive  $M M' M''$ )
    hence  $\Gamma \vdash M' : \tau$  using preservation by metis
    hence  $\Gamma \vdash M'' : \tau$  using preservation'  $\langle M' \rightarrow\beta M'' \rangle$  by metis
    thus ?case using progress by metis
  next
qed

```

```

inductive parallel-reduction :: 'a trm  $\Rightarrow$  'a trm  $\Rightarrow$  bool ( $- \gg -$ ) where
  refl:  $A \gg A$ 
| beta:  $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (App (Fn x T A) B) \gg (A'[x ::= B'])$ 
| eta:  $A \gg A' \Longrightarrow (Fn x T A) \gg (Fn x T A')$ 
| app:  $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (App A B) \gg (App A' B')$ 
| pair:  $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (Pair A B) \gg (Pair A' B')$ 
| fst1:  $P \gg P' \Longrightarrow (Fst P) \gg (Fst P')$ 
| fst2:  $A \gg A' \Longrightarrow (Fst (Pair A B)) \gg A'$ 
| snd1:  $P \gg P' \Longrightarrow (Snd P) \gg (Snd P')$ 
| snd2:  $B \gg B' \Longrightarrow (Snd (Pair A B)) \gg B'$ 

```

```

lemma parallel-reduction-prm:
  assumes  $A \gg A'$ 
  shows  $(\pi \cdot A) \gg (\pi \cdot A')$ 
using assms
  apply induction
  apply (rule parallel-reduction.refl)
  apply (metis parallel-reduction.beta subst-prm trm-prm-simp(3, 4))
  apply (metis parallel-reduction.eta trm-prm-simp(4))
  apply (metis parallel-reduction.app trm-prm-simp(3))
  apply (metis parallel-reduction.pair trm-prm-simp(5))
  apply (metis parallel-reduction.fst1 trm-prm-simp(6))
  apply (metis parallel-reduction.fst2 trm-prm-simp(5, 6))
  apply (metis parallel-reduction.snd1 trm-prm-simp(7))
  apply (metis parallel-reduction.snd2 trm-prm-simp(5, 7))
done

```

```

lemma parallel-reduction-fvs:
  assumes  $A \gg A'$ 
  shows  $fvs A' \subseteq fvs A$ 
using assms proof(induction)
  case (refl A)
    thus ?case by auto
  next
  case (beta A A' B B' x T)
    have  $*:fvs (App (Fn x T A) B) = fvs A - \{x\} \cup fvs B$  using fvs-simp(3, 4)
by metis
    have  $fvs (A'[x ::= B']) \subseteq fvs A' - \{x\} \cup fvs B'$  using subst-fvs.
    also have  $\dots \subseteq fvs A - \{x\} \cup fvs B$  using beta.IH by auto
    finally show ?case using fvs-simp(3, 4) by metis
  next
  case (eta A A' x T)
    thus ?case using fvs-simp(4) Un-Diff subset-Un-eq by metis
  next
  case (app A A' B B')
    thus ?case using fvs-simp(3) Un-mono by metis
  next
  case (pair A A' B B')
    thus ?case using fvs-simp(5) Un-mono by metis
  next
  case (fst1 P P')
    thus ?case using fvs-simp(6) by force
  next
  case (fst2 A A' B)
    thus ?case using fvs-simp(5, 6) by force
  next
  case (snd1 P P')
    thus ?case using fvs-simp(7) by force
  next
  case (snd2 B B' A)

```

```

    thus ?case using fvs-simp(5, 7) by force
  next
qed

inductive-cases parallel-reduction-unitE': Unit >> A
lemma parallel-reduction-unitE:
  assumes Unit >> A
  shows A = Unit
using assms
  apply (rule parallel-reduction-unitE'[where A=A])
  apply blast
  apply (auto simp add: unit-not-app unit-not-fn unit-not-pair unit-not-fst unit-not-snd)
done

inductive-cases parallel-reduction-varE': (Var x) >> A
lemma parallel-reduction-varE:
  assumes (Var x) >> A
  shows A = Var x
using assms
  apply (rule parallel-reduction-varE'[where x=x and A=A])
  apply blast
  apply (auto simp add: var-not-app var-not-fn var-not-pair var-not-fst var-not-snd)
done

inductive-cases parallel-reduction-fnE': (Fn x T A) >> X
lemma parallel-reduction-fnE:
  assumes (Fn x T A) >> X
  shows X = Fn x T A  $\vee$  ( $\exists A'$ . (A >> A')  $\wedge$  X = Fn x T A')
using assms proof(induction rule: parallel-reduction-fnE'[where x=x and T=T
and A=A and X=X])
  case (4 B B' y S)
    from this consider x = y  $\wedge$  T = S  $\wedge$  A = B | x  $\neq$  y  $\wedge$  T = S  $\wedge$  x  $\notin$  fvs B
   $\wedge$  A = [x  $\leftrightarrow$  y]  $\cdot$  B
    using trm-simp(4) by metis
  thus ?case proof(cases)
    case 1
      hence x = y T = S A = B by auto
      thus ?thesis using 4 by metis
    next
      case 2
        hence x  $\neq$  y T = S x  $\notin$  fvs B A = [x  $\leftrightarrow$  y]  $\cdot$  B by auto
        hence x  $\notin$  fvs B' A >> ([x  $\leftrightarrow$  y]  $\cdot$  B')
          using parallel-reduction-fvs parallel-reduction-prm (B >> B') by auto
        thus ?thesis using fn-eq (X = Fn y S B') (x  $\neq$  y) (T = S) by metis
      next
        qed
    next
      qed
  qed (
    metis assms,

```

*blast,*  
*metis app-not-fn,*  
*metis app-not-fn,*  
*metis fn-not-pair,*  
*metis fn-not-fst,*  
*metis fn-not-fst,*  
*metis fn-not-snd,*  
*metis fn-not-snd*  
)

**inductive-cases** *parallel-reduction-redexE'*:  $(App (Fn x T A) B) \gg X$

**lemma** *parallel-reduction-redexE*:

**assumes**  $(App (Fn x T A) B) \gg X$

**shows**

$(X = App (Fn x T A) B) \vee$

$(\exists A' B'. (A \gg A') \wedge (B \gg B') \wedge X = (A'[x ::= B'])) \vee$

$(\exists A' B'. ((Fn x T A) \gg (Fn x T A')) \wedge (B \gg B') \wedge X = (App (Fn x T A') B'))$

**using** *assms* **proof**(*induction rule: parallel-reduction-redexE'*[**where**  $x=x$  **and**  $T=T$  **and**  $A=A$  **and**  $B=B$  **and**  $X=X$ ])

**case**  $(5 C C' D D')$

**from**  $\langle App (Fn x T A) B = App C D \rangle$  **have**  $C: C = Fn x T A$  **and**  $D: D = B$

**by** (*metis trm-simp(2)*, *metis trm-simp(3)*)

**from**  $C$  **and**  $\langle C \gg C' \rangle$  **obtain**  $A'$  **where**  $C': C' = Fn x T A'$

**using** *parallel-reduction-fnE* **by** *metis*

**thus** *?thesis* **using**  $C C' D \langle C \gg C' \rangle \langle D \gg D' \rangle \langle X = App C' D' \rangle$  **by** *metis*

**next**

**case**  $(3 C C' D D' y S)$

**from**  $\langle App (Fn x T A) B = App (Fn y S C) D \rangle$  **have**  $Fn x T A = Fn y S C$

**and**  $B: B = D$

**by** (*metis trm-simp(2)*, *metis trm-simp(3)*)

**from** *this* **consider**

$x = y \wedge T = S \wedge A = C$

$| x \neq y \wedge T = S \wedge A = [x \leftrightarrow y] \cdot C \wedge x \notin fvs C$

**using** *trm-simp(4)* **by** *metis*

**thus** *?case* **proof**(*cases*)

**case** 1

**thus** *?thesis* **using**  $\langle C \gg C' \rangle \langle X = (C'[y ::= D']) \rangle \langle D \gg D' \rangle B$  **by** *metis*

**next**

**case** 2

**hence**  $x \neq y T = S$  **and**  $A: A = [x \leftrightarrow y] \cdot C \wedge x \notin fvs C$  **by** *auto*

**have**  $x \notin fvs C'$  **using** *parallel-reduction-fvs*  $\langle x \notin fvs C \rangle \langle C \gg C' \rangle$  **by**

*force*

**have**  $A \gg ([x \leftrightarrow y] \cdot C')$

**using** *parallel-reduction-prm*  $\langle C \gg C' \rangle A$  **by** *metis*

**moreover** **have**  $X = (([x \leftrightarrow y] \cdot C')[x ::= D'])$

**using**  $\langle X = (C'[y ::= D']) \rangle$  *subst-swp*  $\langle x \notin fvs C' \rangle$  **by** *metis*

```

      ultimately show ?thesis using ⟨D >> D'⟩ B by metis
    next
    qed
  next
qed (
  metis assms,
  blast,
  metis app-not-fn,
  metis app-not-pair,
  metis app-not-fst,
  metis app-not-fst,
  metis app-not-snd,
  metis app-not-snd
)

inductive-cases parallel-reduction-nonredexE': (App A B) >> X
lemma parallel-reduction-nonredexE:
  assumes (App A B) >> X and  $\bigwedge x T A'. A \neq Fn\ x\ T\ A'$ 
  shows  $\exists A' B'. (A >> A') \wedge (B >> B') \wedge X = (App\ A'\ B')$ 
using assms proof(induction rule: parallel-reduction-nonredexE'[where A=A and
B=B and X=X])
  case (5 C C' D D')
    hence A = C B = D using trm-simp(2, 3) by auto
    thus ?case using ⟨C >> C'⟩ ⟨D >> D'⟩ ⟨X = App C' D'⟩ by metis
  next
qed (
  metis assms(1),
  metis parallel-reduction.refl,
  metis trm-simp(2, 3) assms(2),
  metis app-not-fn,
  metis app-not-pair,
  metis app-not-fst,
  metis app-not-fst,
  metis app-not-snd,
  metis app-not-snd
)

inductive-cases parallel-reduction-pairE': (Pair A B) >> X
lemma parallel-reduction-pairE:
  assumes (Pair A B) >> X
  shows  $\exists A' B'. (A >> A') \wedge (B >> B') \wedge (X = Pair\ A'\ B')$ 
using assms proof(induction rule: parallel-reduction-pairE'[where A=A and B=B
and X=X])
  case 2
    thus ?case using parallel-reduction.refl by blast
  next
  case (6 A A' B B')
    thus ?case using parallel-reduction.pair trm-simp(5, 6) by fastforce
  next

```



```

qed (
  metis assms,
  metis app-not-pair,
  metis fn-not-pair,
  metis app-not-pair,
  metis pair-not-fst,
  metis pair-not-fst,
  metis pair-not-snd,
  metis pair-not-snd
)

inductive-cases parallel-reduction-fstE': (Fst P) >> X
lemma parallel-reduction-fstE:
  assumes (Fst P) >> X
  shows ( $\exists P'. (P >> P') \wedge X = (Fst P')$ )  $\vee$  ( $\exists A A' B. (P = Pair A B) \wedge (A >> A') \wedge X = A'$ )
using assms proof(induction rule: parallel-reduction-fstE'[where P=P and X=X])
  case (7 P P')
    thus ?case using parallel-reduction.fst1 trm-simp(7) by metis
  next
  case (8 A B)
    thus ?case using parallel-reduction.fst2 trm-simp(7) by metis
  next
qed (
  metis assms,
  insert parallel-reduction.refl, metis,
  metis app-not-fst,
  metis fn-not-fst,
  metis app-not-fst,
  metis pair-not-fst,
  metis fst-not-snd,
  metis fst-not-snd
)

inductive-cases parallel-reduction-sndE': (Snd P) >> X
lemma parallel-reduction-sndE:
  assumes (Snd P) >> X
  shows ( $\exists P'. (P >> P') \wedge X = (Snd P')$ )  $\vee$  ( $\exists A B B'. (P = Pair A B) \wedge (B >> B') \wedge X = B'$ )
using assms proof(induction rule: parallel-reduction-sndE'[where P=P and X=X])
  case (9 P P')
    thus ?case using parallel-reduction.snd1 trm-simp(8) by metis
  next
  case (10 A B)
    thus ?case using parallel-reduction.snd2 trm-simp(8) by metis
  next
qed (
  metis assms,
  insert parallel-reduction.refl, metis,

```

```

metis app-not-snd,
metis fn-not-snd,
metis app-not-snd,
metis pair-not-snd,
metis fst-not-snd,
metis fst-not-snd
)

```

**lemma** *parallel-reduction-subst-inner*:

```

assumes  $M \gg M'$ 
shows  $X[z ::= M] \gg (X[z ::= M'])$ 
using assms proof(induction  $X$  rule: trm-strong-induct[where  $S = \{z\} \cup fvs\ M \cup fvs\ M'$ ])
show finite ( $\{z\} \cup fvs\ M \cup fvs\ M'$ ) using fvs-finite by auto

```

```

case 1
  thus ?case using subst-simp-unit parallel-reduction.refl by metis
next
case (2  $x$ )
  thus ?case by(cases  $x = z$ , metis subst-simp-var1, metis subst-simp-var2
parallel-reduction.refl)
next
case (3  $A\ B$ )
  thus ?case using subst-simp-app parallel-reduction.app by metis
next
case (4  $x\ T\ A$ )
  hence  $x \neq z$   $x \notin fvs\ M$   $x \notin fvs\ M'$  by auto
  thus ?case using 4 subst-simp-fn parallel-reduction.eta by metis
next
case (5  $A\ B$ )
  thus ?case using subst-simp-pair parallel-reduction.pair by metis
next
case (6  $P$ )
  thus ?case using subst-simp-fst parallel-reduction.fst1 by metis
next
case (7  $P$ )
  thus ?case using subst-simp-snd parallel-reduction.snd1 by metis
next
qed

```

**lemma** *parallel-reduction-subst*:

```

assumes  $X \gg X'$   $M \gg M'$ 
shows  $X[z ::= M] \gg (X'[z ::= M'])$ 
using assms proof(induction  $X$  arbitrary:  $X'$  rule: trm-strong-depth-induct[where  $S = \{z\} \cup fvs\ M \cup fvs\ M'$ ])
show finite ( $\{z\} \cup fvs\ M \cup fvs\ M'$ ) using fvs-finite by auto
next

```

```

case 1

```

hence  $X' = \text{Unit}$  **using** *parallel-reduction-unitE* **by** *metis*  
 thus  $?case$  **using** *parallel-reduction.refl subst-simp-unit* **by** *metis*  
 next  
 case (2  $x$ )  
 hence  $X' = \text{Var } x$  **using** *parallel-reduction-varE* **by** *metis*  
 thus  $?case$  **using** *parallel-reduction-subst-inner*  $\langle M \gg M' \rangle$  **by** *metis*  
 next  
 case (3  $C D$ )  
 consider  $\exists x T A. C = Fn\ x\ T\ A \mid \nexists x T A. C = Fn\ x\ T\ A$  **by** *metis*  
 thus  $?case$  **proof**(*cases*)  
 case 1  
 from *this* **obtain**  $x\ T\ A$  **where**  $C: C = Fn\ x\ T\ A$  **by** *auto*  
 have  $\text{depth } C < \text{depth } (\text{App } C\ D)$   $\text{depth } D < \text{depth } (\text{App } C\ D)$   
 using *depth-app* **by** *auto*  
  
 consider  
 $X' = \text{App } (Fn\ x\ T\ A)\ D$   
 $\mid \exists A' D'. ((Fn\ x\ T\ A) \gg (Fn\ x\ T\ A')) \wedge (D \gg D') \wedge X' = (\text{App } (Fn\ x\ T\ A')\ D')$   
 $\mid \exists A' D'. (A \gg A') \wedge (D \gg D') \wedge X' = (A'[x ::= D'])$   
 using *parallel-reduction-redexE*  $\langle (\text{App } C\ D) \gg X' \rangle$   $C$  **by** *metis*  
 thus  $?thesis$  **proof**(*cases*)  
 case 1  
 thus  $?thesis$  **using** *parallel-reduction-subst-inner*  $\langle M \gg M' \rangle$   $C$  **by** *metis*  
 next  
 case 2  
 from *this* **obtain**  $A' D'$   
 where  $(Fn\ x\ T\ A) \gg (Fn\ x\ T\ A')\ D \gg D'$  **and**  $X': X' = \text{App } (Fn\ x\ T\ A')\ D'$   
 by *auto*  
 have \*:  $((Fn\ x\ T\ A)[z ::= M]) \gg ((Fn\ x\ T\ A')[z ::= M'])$   
 using 3.IH  $\langle \text{depth } C < \text{depth } (\text{App } C\ D) \rangle$   $C \langle (Fn\ x\ T\ A) \gg (Fn\ x\ T\ A') \rangle \langle M \gg M' \rangle$   
 by *metis*  
 have \*\*:  $(D[z ::= M]) \gg (D'[z ::= M'])$   
 using 3.IH  $\langle \text{depth } D < \text{depth } (\text{App } C\ D) \rangle$   $\langle D \gg D' \rangle \langle M \gg M' \rangle$   
 by *metis*  
  
 have  $(\text{App } C\ D)[z ::= M] = \text{App } ((Fn\ x\ T\ A)[z ::= M])\ (D[z ::= M])$   
 using *subst-simp-app*  $C$  **by** *metis*  
 moreover have  $\dots \gg (\text{App } ((Fn\ x\ T\ A')[z ::= M'])\ (D'[z ::= M']))$   
 using \* \*\* *parallel-reduction.app* **by** *metis*  
 moreover have  $\dots = ((\text{App } (Fn\ x\ T\ A')\ D')[z ::= M'])$   
 using *subst-simp-app* **by** *metis*  
 moreover have  $\dots = (X'[z ::= M'])$   
 using  $X'$  **by** *metis*  
 ultimately show  $?thesis$  **by** *metis*  
 next  
 case 3

```

from this obtain  $A' D'$  where  $A \gg A' D \gg D'$  and  $X': X' = (A'[x$ 
 $::= D']$ 
  by auto

  have  $\text{depth } A < \text{depth } (\text{App } C D)$ 
    using  $C \text{ depth-app depth-fn dual-order.strict-trans}$  by fastforce

  have  $\text{finite } (\{z\} \cup \text{fvs } M \cup \text{fvs } M' \cup \text{fvs } A')$  using fvs-finite by auto
  from this obtain  $y$ 
    where  $y \notin \{z\} \cup \text{fvs } M \cup \text{fvs } M' \cup \text{fvs } A'$  and  $C: C = \text{Fn } y T ([y$ 
 $\leftrightarrow x] \cdot A)$ 
    using fresh-fn  $C$  by metis
    hence  $y \neq z \ y \notin \text{fvs } M \ y \notin \text{fvs } M' \ y \notin \text{fvs } A'$  by auto
    have  $([y \leftrightarrow x] \cdot A) \gg ([y \leftrightarrow x] \cdot A')$  using parallel-reduction-prm  $\langle A$ 
 $\gg A' \rangle$  by metis
    hence  $*$ :  $([y \leftrightarrow x] \cdot A)[z ::= M] \gg (([y \leftrightarrow x] \cdot A')[z ::= M'])$ 
      using  $\text{3.IH } \langle \text{depth } A < \text{depth } (\text{App } C D) \rangle \text{ depth-prm}$ 
      using  $\langle ([y \leftrightarrow x] \cdot A) \gg ([y \leftrightarrow x] \cdot A') \rangle \langle M \gg M' \rangle$  by metis
    have  $**$ :  $(D[z ::= M]) \gg (D'[z ::= M'])$ 
      using  $\text{3.IH } \langle \text{depth } D < \text{depth } (\text{App } C D) \rangle \langle D \gg D' \rangle \langle M \gg M' \rangle$ 
      by metis

    have  $(\text{App } C D)[z ::= M] = (\text{App } ((\text{Fn } y T ([y \leftrightarrow x] \cdot A)))[z ::= M])$ 
 $(D[z ::= M])$ 
      using  $C \text{ subst-simp-app}$  by metis
    moreover have  $\dots = (\text{App } (\text{Fn } y T (([y \leftrightarrow x] \cdot A)[z ::= M])) (D[z ::=$ 
 $M]))$ 
      using  $\langle y \neq z \rangle \langle y \notin \text{fvs } M \rangle \text{subst-simp-fn}$  by metis
    moreover have  $\dots \gg (([y \leftrightarrow x] \cdot A')[z ::= M'] [y ::= D'[z ::= M']])$ 
      using parallel-reduction.beta * ** by metis
    moreover have  $\dots = (([y \leftrightarrow x] \cdot A')[y ::= D'] [z ::= M'])$ 
      using barendregt  $\langle y \neq z \rangle \langle y \notin \text{fvs } M' \rangle$  by metis
    moreover have  $\dots = (A'[x ::= D'] [z ::= M'])$ 
      using subst-swp  $\langle y \notin \text{fvs } A' \rangle$  by metis
    moreover have  $\dots = (X'[z ::= M'])$  using  $X'$  by metis
    ultimately show  $?thesis$  by metis
  next
  qed
next
case 2
  from this obtain  $C' D'$  where  $C \gg C' D \gg D'$  and  $X': X' = \text{App}$ 
 $C' D'$ 
    using parallel-reduction-nonredexE  $\langle (\text{App } C D) \gg X' \rangle$  by metis

  have  $\text{depth } C < \text{depth } (\text{App } C D) \ \text{depth } D < \text{depth } (\text{App } C D)$ 
    using depth-app by auto
  hence  $*$ :  $(C[z ::= M]) \gg (C'[z ::= M'])$  and  $**$ :  $(D[z ::= M]) \gg (D'[z$ 
 $::= M'])$ 
    using  $\text{3.IH } \langle C \gg C' \rangle \langle D \gg D' \rangle \langle M \gg M' \rangle$  by metis+

```

**have**  $(App\ C\ D)[z ::= M] = App\ (C[z ::= M])\ (D[z ::= M])$   
**using** *subst-simp-app* **by** *metis*  
**moreover have**  $\dots \gg (App\ (C'[z ::= M'])\ (D'[z ::= M']))$   
**using** *parallel-reduction.app* **\*\* by** *metis*  
**moreover have**  $\dots = ((App\ C'\ D')[z ::= M'])$   
**using** *subst-simp-app* **by** *metis*  
**moreover have**  $\dots = (X'[z ::= M'])$  **using**  $X'$  **by** *metis*  
**ultimately show** *?thesis* **by** *metis*  
**next**  
**qed**  
**next**  
**case**  $(\not\vdash x\ T\ A)$   
**hence**  $x \neq z\ x \notin fvs\ M\ x \notin fvs\ M'$   
**by** *auto*  
  
**from**  $\langle Fn\ x\ T\ A \rangle \gg X'$  **consider**  
 $X' = Fn\ x\ T\ A$   
 $|\exists A'. (A \gg A') \wedge X' = Fn\ x\ T\ A'$  **using** *parallel-reduction-fnE* **by** *metis*  
**thus** *?case* **proof**(*cases*)  
**case** 1  
**thus** *?thesis* **using** *parallel-reduction-subst-inner*  $\langle M \gg M' \rangle$  **by** *metis*  
**next**  
**case** 2  
**from** *this* **obtain**  $A'$  **where**  $A \gg A'$  **and**  $X': X' = Fn\ x\ T\ A'$  **by** *auto*  
  
**hence**  $*$ :  $(A[z ::= M]) \gg (A'[z ::= M'])$   
**using** 4.*IH* *depth-fn*  $\langle A \gg A' \rangle \langle M \gg M' \rangle$  **by** *metis*  
  
**have**  $((Fn\ x\ T\ A)[z ::= M]) = (Fn\ x\ T\ (A[z ::= M]))$   
**using** *subst-simp-fn*  $\langle x \neq z \rangle \langle x \notin fvs\ M \rangle$  **by** *metis*  
**moreover have**  $\dots \gg (Fn\ x\ T\ (A'[z ::= M']))$   
**using** *parallel-reduction.eta* **\*** **by** *metis*  
**moreover have**  $\dots = ((Fn\ x\ T\ A')[z ::= M'])$   
**using** *subst-simp-fn*  $\langle x \neq z \rangle \langle x \notin fvs\ M' \rangle$  **by** *metis*  
**moreover have**  $\dots = (X'[z ::= M'])$   
**using**  $X'$  **by** *metis*  
**ultimately show** *?thesis* **by** *metis*  
**next**  
**qed**  
**next**  
**case**  $(\not\vdash A\ B)$   
**from**  $\langle Pair\ A\ B \rangle \gg X'$  **consider**  
 $X' = Pair\ A\ B$   
 $|\exists A'\ B'. (A \gg A') \wedge (B \gg B') \wedge X' = Pair\ A'\ B'$   
**using** *parallel-reduction-pairE* **by** *metis*  
**thus** *?case* **proof**(*cases*)  
**case** 1  
**thus** *?thesis* **using** *parallel-reduction-subst-inner*  $\langle M \gg M' \rangle$  **by** *metis*

```

next
case 2
  from this obtain A' B' where A >> A' B >> B' and X': X' = Pair A'
  B' by auto

  have *: (A[z ::= M]) >> (A'[z ::= M']) and **: (B[z ::= M]) >> (B'[z
  ::= M'])
  using 5.IH ⟨A >> A'⟩ ⟨B >> B'⟩ ⟨M >> M'⟩ by (metis depth-pair(1),
  metis depth-pair(2))

  have (Pair A B)[z ::= M] = (Pair (A[z ::= M]) (B[z ::= M]))
  using subst-simp-pair by metis
  moreover have ... >> (Pair (A'[z ::= M']) (B'[z ::= M']))
  using parallel-reduction.pair * ** by metis
  moreover have ... = ((Pair A' B')[z ::= M'])
  using subst-simp-pair by metis
  moreover have ... = (X'[z ::= M']) using X' by metis
  ultimately show ?thesis by metis
next
qed
next
case (6 P)
  from ⟨(Fst P) >> X'⟩ consider
  ∃ P'. (P >> P') ∧ X' = Fst P'
  | ∃ A A' B. P = Pair A B ∧ (A >> A') ∧ X' = A'
  using parallel-reduction.fstE by metis
  thus ?case proof(cases)
  case 1
    from this obtain P' where P >> P' and X': X' = Fst P' by auto

    have *: (P[z ::= M]) >> (P'[z ::= M'])
    using 6.IH depth-fst ⟨P >> P'⟩ ⟨M >> M'⟩ by metis

    have (Fst P)[z ::= M] = Fst (P[z ::= M])
    using subst-simp-fst by metis
    moreover have ... >> (Fst (P'[z ::= M']))
    using parallel-reduction.fst1 * by metis
    moreover have ... = ((Fst P')[z ::= M'])
    using subst-simp-fst by metis
    moreover have ... = (X'[z ::= M']) using X' by metis
    ultimately show ?thesis by metis
  next
  case 2
    from this obtain A A' B where P: P = Pair A B A >> A' and X': X'
    = A' by auto

    have depth A < depth (Fst P)
    using P depth-fst depth-pair dual-order.strict-trans by fastforce
    hence *: (A[z ::= M]) >> (A'[z ::= M'])

```

```

    using 6.IH  $\langle A \gg A' \rangle \langle M \gg M' \rangle$  by metis

  have (Fst P)[z ::= M] = (Fst (Pair (A[z ::= M]) (B[z ::= M])))
    using P subst-simp-fst subst-simp-pair by metis
  moreover have ...  $\gg (A'[z ::= M'])$ 
    using parallel-reduction.fst2 * by metis
  moreover have ... = (X'[z ::= M'])
    using X' by metis
  ultimately show ?thesis by metis
next
qed
next
case ( $\neg P$ )
  from  $\langle (Snd P) \gg X' \rangle$  consider
     $\exists P'. (P \gg P') \wedge X' = Snd P'$ 
  |  $\exists A B B'. P = Pair A B \wedge (B \gg B') \wedge X' = B'$ 
    using parallel-reduction-sndE by metis
  thus ?case proof(cases)
  case 1
    from this obtain P' where  $P \gg P'$  and  $X': X' = Snd P'$  by auto

    have *:  $(P[z ::= M]) \gg (P'[z ::= M'])$ 
      using 7.IH depth-snd  $\langle P \gg P' \rangle \langle M \gg M' \rangle$  by metis

    have (Snd P)[z ::= M] = Snd (P[z ::= M])
      using subst-simp-snd by metis
    moreover have ...  $\gg (Snd (P'[z ::= M']))$ 
      using parallel-reduction.snd1 * by metis
    moreover have ... =  $((Snd P')[z ::= M'])$ 
      using subst-simp-snd by metis
    moreover have ... =  $(X'[z ::= M'])$  using X' by metis
    ultimately show ?thesis by metis
  next
  case 2
    from this obtain A B B' where  $P: P = Pair A B \wedge (B \gg B')$  and  $X': X' = B'$  by auto

    have depth B < depth (Snd P)
      using P depth-snd depth-pair dual-order.strict-trans by fastforce
    hence *:  $(B[z ::= M]) \gg (B'[z ::= M'])$ 
      using 7.IH  $\langle B \gg B' \rangle \langle M \gg M' \rangle$  by metis

    have (Snd P)[z ::= M] = (Snd (Pair (A[z ::= M]) (B[z ::= M])))
      using P subst-simp-snd subst-simp-pair by metis
    moreover have ...  $\gg (B'[z ::= M'])$ 
      using parallel-reduction.snd2 * by metis
    moreover have ... =  $(X'[z ::= M'])$ 
      using X' by metis
    ultimately show ?thesis by metis

```

next  
qed  
next  
qed

**inductive complete-development** :: 'a trm  $\Rightarrow$  'a trm  $\Rightarrow$  bool (- >>> -) **where**  
*unit*: Unit >>> Unit  
| *var*: (Var x) >>> (Var x)  
| *beta*:  $\llbracket A \ggg A'; B \ggg B' \rrbracket \Longrightarrow (App (Fn x T A) B) \ggg (A'[x ::= B'])$   
| *eta*:  $A \ggg A' \Longrightarrow (Fn x T A) \ggg (Fn x T A')$   
| *app*:  $\llbracket A \ggg A'; B \ggg B'; (\bigwedge x T M. A \neq Fn x T M) \rrbracket \Longrightarrow (App A B) \ggg (App A' B')$   
| *pair*:  $\llbracket A \ggg A'; B \ggg B' \rrbracket \Longrightarrow (Pair A B) \ggg (Pair A' B')$   
| *fst1*:  $\llbracket P \ggg P'; (\bigwedge A B. P \neq Pair A B) \rrbracket \Longrightarrow (Fst P) \ggg (Fst P')$   
| *fst2*:  $A \ggg A' \Longrightarrow (Fst (Pair A B)) \ggg A'$   
| *snd1*:  $\llbracket P \ggg P'; (\bigwedge A B. P \neq Pair A B) \rrbracket \Longrightarrow (Snd P) \ggg (Snd P')$   
| *snd2*:  $B \ggg B' \Longrightarrow (Snd (Pair A B)) \ggg B'$

**lemma not-fn-prm**:

**assumes**  $\bigwedge x T M. A \neq Fn x T M$   
**shows**  $\pi \cdot A \neq Fn x T M$

**proof**(rule classical)

**fix** x T M

**obtain**  $\pi'$  **where** \*:  $\pi' = prm\text{-}inv \pi$  **by** auto

**assume**  $\neg(\pi \cdot A \neq Fn x T M)$

**hence**  $\pi \cdot A = Fn x T M$  **by** blast

**hence**  $\pi' \cdot (\pi \cdot A) = \pi' \cdot Fn x T M$  **by** fastforce

**hence**  $(\pi' \diamond \pi) \cdot A = \pi' \cdot Fn x T M$

**using** trm-prm-apply-compose **by** metis

**hence**  $A = \pi' \cdot Fn x T M$

**using** \* prm-inv-compose trm-prm-apply-id **by** metis

**hence**  $A = Fn (\pi' \$ x) T (\pi' \cdot M)$  **using** trm-prm-simp(4) **by** metis

**hence** False **using** assms **by** blast

**thus** ?thesis **by** blast

qed

**lemma not-pair-prm**:

**assumes**  $\bigwedge A B. P \neq Pair A B$

**shows**  $\pi \cdot P \neq Pair A B$

**proof**(rule classical)

**fix** A B

**obtain**  $\pi'$  **where** \*:  $\pi' = prm\text{-}inv \pi$  **by** auto

**assume**  $\neg(\pi \cdot P \neq Pair A B)$

**hence**  $\pi \cdot P = Pair A B$  **by** blast

**hence**  $\pi' \cdot \pi \cdot P = \pi' \cdot (Pair A B)$  **by** fastforce

**hence**  $(\pi' \diamond \pi) \cdot P = \pi' \cdot (Pair A B)$

**using** trm-prm-apply-compose **by** metis

**hence**  $P = \pi' \cdot (Pair A B)$

**using** \* prm-inv-compose trm-prm-apply-id **by** metis



hence  $P = \text{Pair } (\pi' \cdot A) (\pi' \cdot B)$  **using** *trm-prm-simp(5)* **by** *metis*  
 hence *False* **using** *assms* **by** *blast*  
 thus *?thesis* **by** *blast*  
**qed**

**lemma** *complete-development-prm*:  
 assumes  $A \ggg A'$   
 shows  $(\pi \cdot A) \ggg (\pi \cdot A')$   
**using** *assms* **proof**(*induction*)  
 case *unit*  
   **thus** *?case* **using** *complete-development.unit trm-prm-simp(1)* **by** *metis*  
 next  
 case (*var x*)  
   **thus** *?case* **using** *complete-development.var trm-prm-simp(2)* **by** *metis*  
 next  
 case (*beta A A' B B' x T*)  
   **thus** *?case* **using** *complete-development.beta subst-prm trm-prm-simp(3, 4)* **by** *metis*  
*metis*  
 next  
 case (*eta A A' x T*)  
   **thus** *?case* **using** *complete-development.eta trm-prm-simp(4)* **by** *metis*  
 next  
 case (*app A A' B B'*)  
   **thus** *?case* **using** *complete-development.app* **by** (*simp add: trm-prm-simp(3)*)  
*not-fn-prm*  
 next  
 case (*pair A A' B B'*)  
   **thus** *?case* **using** *complete-development.pair trm-prm-simp(5)* **by** *metis*  
 next  
 case (*fst1 P P'*)  
   **hence**  $\pi \cdot P \neq \text{Pair } A B$  **for**  $A B$  **using** *not-pair-prm* **by** *metis*  
   **thus** *?case* **using** *trm-prm-simp(6) fst1.IH complete-development.fst1* **by** *metis*  
 next  
 case (*fst2 A A' B*)  
   **thus** *?case* **using** *trm-prm-simp(5, 6) complete-development.fst2* **by** *metis*  
 next  
 case (*snd1 P P'*)  
   **hence**  $\pi \cdot P \neq \text{Pair } A B$  **for**  $A B$  **using** *not-pair-prm* **by** *metis*  
   **thus** *?case* **using** *trm-prm-simp(7) snd1.IH complete-development.snd1* **by** *metis*  
*metis*  
 next  
 case (*snd2 B B' A*)  
   **thus** *?case* **using** *trm-prm-simp(5, 7) complete-development.snd2* **by** *metis*  
 next  
**qed**

**lemma** *complete-development-fvs*:  
 assumes  $A \ggg A'$   
 shows  $\text{fvs } A' \subseteq \text{fvs } A$

```

using assms proof(induction)
  case unit
    thus ?case using fvs-simp by auto
  next
  case (var x)
    thus ?case using fvs-simp by auto
  next
  case (beta A A' B B' x T)
    have  $fvs (A'[x ::= B']) \subseteq fvs A' - \{x\} \cup fvs B'$  using subst-fvs.
    also have  $\dots \subseteq fvs A - \{x\} \cup fvs B$  using beta.IH by auto
    also have  $\dots \subseteq fvs (Fn\ x\ T\ A) \cup fvs B$  using fvs-simp(4) subset-refl by force
    also have  $\dots \subseteq fvs (App\ (Fn\ x\ T\ A)\ B)$  using fvs-simp(3) subset-refl by force
    finally show ?case.
  next
  case (eta A A' x T)
    thus ?case using fvs-simp(4) using Un-Diff subset-Un-eq by (metis (no-types,
lifting))
  next
  case (app A A' B B')
    thus ?case using fvs-simp(3) Un-mono by metis
  next
  case (pair A A' B B')
    thus ?case using fvs-simp(5) Un-mono by metis
  next
  case (fst1 P P')
    thus ?case using fvs-simp(6) by force
  next
  case (fst2 A A' B)
    thus ?case by (simp add: fvs-simp(5, 6) sup.coboundedI1)
  next
  case (snd1 P P')
    thus ?case using fvs-simp(7) by fastforce
  next
  case (snd2 B B' A)
    thus ?case using fvs-simp(5, 7) by fastforce
  next
qed

```

**lemma** *complete-development-fnE*:

**assumes**  $(Fn\ x\ T\ A) \gggg\ X$

**shows**  $\exists A'. (A \gggg\ A') \wedge X = Fn\ x\ T\ A'$

**using** *assms* **proof**(*cases*)

**case** (*eta B B' y S*)

**hence**  $T = S$  **using** *trm-simp(4)* **by** *metis*

**from** *eta* **consider**  $x = y \wedge A = B \mid x \neq y \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot B$

**using** *trm-simp(4)* **by** *metis*

**thus** ?*thesis* **proof**(*cases*)

**case** *1*

**hence**  $x = y$  **and**  $A = B$  **by** *auto*

```

    obtain A' where A' = B' by auto
    hence A >>> A' and X = Fn x T A' using eta (A = B) (x = y) (T =
S) by auto
    thus ?thesis by auto
  next
  case 2
  hence x ≠ y x ∉ fvs B and A: A = [x ↔ y] · B by auto
  hence x ∉ fvs B' using (B >>> B') complete-development-fvs by auto
  obtain A' where A': A' = [x ↔ y] · B' by auto
  hence A >>> A' using A (B >>> B') complete-development-prm by auto
  have X = Fn x T A'
    using fn-eq (x ≠ y) (x ∉ fvs B') A' (X = Fn y S B') (T = S) by metis
  thus ?thesis using (A >>> A') by auto
  next
  qed
  next
  qed (
    metis unit-not-fn,
    metis var-not-fn,
    metis app-not-fn,
    metis app-not-fn,
    metis fn-not-pair,
    metis fn-not-fst,
    metis fn-not-fst,
    metis fn-not-snd,
    metis fn-not-snd
  )

```

```

lemma complete-development-pairE:
  assumes (Pair A B) >>> X
  shows ∃ A' B'. (A >>> A') ∧ (B >>> B') ∧ X = Pair A' B'
using assms
  apply cases
  apply (metis unit-not-pair, metis var-not-pair, metis app-not-pair, metis fn-not-pair,
metis app-not-pair)
  apply (metis trm-simp(5, 6))
  apply (metis pair-not-fst, metis pair-not-fst, metis pair-not-snd, metis pair-not-snd)
done

```

```

lemma complete-development-exists:
  shows ∃ X. (A >>> X)
proof(induction A rule: trm-induct)
  case 1
  obtain X :: 'a trm where X = Unit by auto
  hence Unit >>> X using complete-development.unit by metis
  thus ?case by auto
  next
  case (2 x)
  obtain X where X = Var x by auto

```

```

    hence (Var x) >>> X using complete-development.var by metis
    thus ?case by auto
next
case (3 A B)
  from this obtain A' B' where A': A >>> A' and B': B >>> B' by auto
  consider (∃ x T M. A = Fn x T M) | ¬(∃ x T M. A = Fn x T M) by auto
  thus ?case proof(cases)
    case 1
      from this obtain x T M where A: A = Fn x T M by auto
      from this obtain M' where A' = Fn x T M' and M >>> M'
        using complete-development.fnE A' by metis
      obtain X where X = (M'[x ::= B']) by auto
      hence (App A B) >>> X
        using complete-development.beta ⟨M >>> M'⟩ B' A by metis
      thus ?thesis by auto
    next
    case 2
      thus ?thesis using complete-development.app A' B' by metis
    next
  qed
next
case (4 x T A)
  from this obtain A' where A': A >>> A' by auto
  obtain X where X = Fn x T A' by auto
  hence (Fn x T A) >>> X using complete-development.eta A' by metis
  thus ?case by auto
next
case (5 A B)
  thus ?case using complete-development.pair by blast
next
case (6 P)
  consider ∃ A B. P = Pair A B | ∄ A B. P = Pair A B by auto
  thus ?case proof(cases)
    case 1
      from this obtain A B X where P = Pair A B P >>> X using 6 by
auto
      from this obtain A' B' where A >>> A' B >>> B' X = Pair A' B'
        using complete-development.pairE by metis
      thus ?thesis using complete-development.fst2 ⟨P = Pair A B⟩ by metis
    next
    case 2
      thus ?thesis using complete-development.fst1 6 by blast
    next
  qed
next
case (7 P)
  consider ∃ A B. P = Pair A B | ∄ A B. P = Pair A B by auto
  thus ?case proof(cases)
    case 1

```

```

    from this obtain A B X where P = Pair A B P >>> X using 7 by
auto
    from this obtain A' B' where A >>> A' B >>> B' X = Pair A' B'
      using complete-development-pairE by metis
    thus ?thesis using complete-development.snd2 ⟨P = Pair A B⟩ by metis
next
case 2
  thus ?thesis using complete-development.snd1 7 by blast
next
qed
next
qed

```

lemma complete-development-triangle:

```

assumes A >>> D A >> B
shows B >> D
using assms proof(induction arbitrary: B rule: complete-development.induct)
case unit
  thus ?case using parallel-reduction-unitE parallel-reduction.refl by metis
next
case (var x)
  thus ?case using parallel-reduction-varE parallel-reduction.refl by metis
next
case (beta A A' C C' x T)
  hence A >> A' C >> C' using parallel-reduction.refl by metis+
  from ⟨(App (Fn x T A) C) >> B⟩ consider
    B = App (Fn x T A) C
  | ∃ A'' C''. (A >> A'') ∧ (C >> C'') ∧ B = (A''[x ::= C''])
  | ∃ A'' C''. ((Fn x T A) >> (Fn x T A'')) ∧ (C >> C'') ∧ B = (App (Fn x
T A'') C'')
  using parallel-reduction-redexE by metis
  thus ?case proof(cases)
case 1
  thus ?thesis using parallel-reduction.beta ⟨A >> A'⟩ ⟨C >> C'⟩ by metis
next
case 2
  from this obtain A'' C'' where A >> A'' C >> C'' and B: B = (A''[x
::= C'']) by auto
  hence A'' >> A' C'' >> C' using beta.IH by metis+
  thus ?thesis using B parallel-reduction-subst by metis
next
case 3
  from this obtain A'' C''
  where (Fn x T A) >> (Fn x T A'') C >> C'' and B: B = App (Fn x
T A'') C''
  by auto
  hence C'' >> C' using beta.IH by metis
  have A'' >> A'
  proof -

```

```

thm parallel-reduction-fnE
from  $\langle (Fn\ x\ T\ A) \gg (Fn\ x\ T\ A'') \rangle$  consider
   $Fn\ x\ T\ A = Fn\ x\ T\ A''$ 
  |  $\exists A''' . (A \gg A''') \wedge Fn\ x\ T\ A'' = Fn\ x\ T\ A'''$ 
  using parallel-reduction-fnE by metis
hence  $A \gg A''$  proof(cases)
  case 1
    hence  $A = A''$  using trm-simp(4) by metis
    thus ?thesis using parallel-reduction.refl by metis
  next
  case 2
    from this obtain  $A'''$  where  $A \gg A'''$   $Fn\ x\ T\ A'' = Fn\ x\ T\ A'''$  by
auto
    thus ?thesis using trm-simp(4) by metis
  next
  qed
thus ?thesis using beta.IH by metis
qed
thus ?thesis using  $B \langle C'' \gg C' \rangle$  parallel-reduction.beta by metis
next
qed
next
case (eta  $A\ A'\ x\ T\ B$ )
from this consider
   $B = Fn\ x\ T\ A$ 
  |  $\exists A'' . (A \gg A'') \wedge B = Fn\ x\ T\ A''$  using parallel-reduction-fnE by metis
thus ?case proof(cases)
  case 1
    thus ?thesis using eta.IH parallel-reduction.refl parallel-reduction.eta by
metis
  next
  case 2
    from this obtain  $A''$  where  $A \gg A''$  and  $B = Fn\ x\ T\ A''$  by auto
    thus ?thesis using eta.IH parallel-reduction.eta by metis
  next
  qed
next
case (app  $A\ A'\ C\ C'$ )
from this obtain  $A''\ C''$  where  $A \gg A''$   $C \gg C''$  and  $B : B = App\ A''\ C''$ 
using parallel-reduction-nonredexE by metis
hence  $A'' \gg A'\ C'' \gg C'$  using app.IH by metis+
thus ?case using  $B$  parallel-reduction.app by metis
next
case (pair  $A\ A'\ C\ C'$ )
from  $\langle (Pair\ A\ C) \gg B \rangle$  and parallel-reduction-pairE obtain  $A''\ C''$  where
 $A \gg A''$   $C \gg C''$   $B = Pair\ A''\ C''$  by metis
thus ?case using pair.IH parallel-reduction.pair by metis
next

```

```

case (fst1 P P')
  from this obtain P'' where P >> P'' B = Fst P''
  using parallel-reduction-fstE by blast
  thus ?case using fst1.IH parallel-reduction.fst1 by metis
next
case (fst2 A A' C B)
  from this consider
     $\exists P''. ((Pair\ A\ C) \gg P'') \wedge B = Fst\ P''$ 
  |  $\exists A''. (A \gg A'') \wedge (B = A'')$ 
  using parallel-reduction-fstE[where P=(Pair A C) and X=B] using trm-simp(5)
by metis
  thus ?case proof(cases)
    case 1
      from this obtain P'' where (Pair A C) >> P'' and B = Fst P'' by auto
      from this obtain A'' C'' where P'' = Pair A'' C'' A >> A'' C >> C''
      using parallel-reduction-pairE by metis
      thus ?thesis using fst2 parallel-reduction.fst2  $\langle B = Fst\ P'' \rangle$  by metis
    next
    case 2
      from this obtain A'' where A >> A'' B = A'' by metis
      thus ?thesis using fst2 by metis
  next
qed
next
case (snd1 P P')
  from this obtain P'' where P >> P'' B = Snd P''
  using parallel-reduction-sndE by blast
  thus ?case using snd1.IH parallel-reduction.snd1 by metis
next
case (snd2 C A' A B)
  from this consider
     $\exists P''. ((Pair\ A\ C) \gg P'') \wedge B = Snd\ P''$ 
  |  $\exists C''. (C \gg C'') \wedge (B = C'')$ 
  using parallel-reduction-sndE[where P=(Pair A C) and X=B] using
trm-simp(5, 6) by metis
  thus ?case proof(cases)
    case 1
      from this obtain P'' where (Pair A C) >> P'' and B = Snd P'' by auto
      from this obtain A'' C'' where P'' = Pair A'' C'' A >> A'' C >> C''
      using parallel-reduction-pairE by metis
      thus ?thesis using snd2 parallel-reduction.snd2  $\langle B = Snd\ P'' \rangle$  by metis
    next
    case 2
      from this obtain C'' where C >> C'' B = C'' by metis
      thus ?thesis using snd2 by metis
  next
qed
next
qed

```

**lemma** *parallel-reduction-diamond*:  
**assumes**  $A \gg B \ A \gg C$   
**shows**  $\exists D. (B \gg D) \wedge (C \gg D)$   
**proof** –  
**obtain**  $D$  **where**  $A \gg \gg D$  **using** *complete-development-exists* **by** *metis*  
**hence**  $(B \gg D) \wedge (C \gg D)$  **using** *complete-development-triangle* **assms** **by**  
*auto*  
**thus** *?thesis* **by** *blast*  
**qed**

**inductive** *parallel-reduces* :: '*a trm*  $\Rightarrow$  '*a trm*  $\Rightarrow$  *bool* ( $- \gg^* -$ ) **where**  
*reflexive*:  $A \gg^* A$   
| *transitive*:  $\llbracket A \gg^* A'; A' \gg A'' \rrbracket \Longrightarrow A \gg^* A''$

**lemma** *parallel-reduces-diamond'*:  
**assumes**  $A \gg^* C \ A \gg B$   
**shows**  $\exists D. (B \gg^* D) \wedge (C \gg D)$   
**using** *assms* **proof**(*induction*)  
**case** (*reflexive*  $A$ )  
**thus** *?case* **using** *parallel-reduces.reflexive* **by** *metis*  
**next**  
**case** (*transitive*  $A \ A' \ A''$ )  
**from** *this* **obtain**  $C$  **where**  $B \gg^* C \ A' \gg C$  **by** *metis*  
**from**  $\langle A' \gg C \rangle \langle A' \gg A'' \rangle$  **obtain**  $D$  **where**  $C \gg D \ A'' \gg D$   
**using** *parallel-reduction-diamond* **by** *metis*  
**thus** *?case* **using** *parallel-reduces.transitive*  $\langle B \gg^* C \rangle$  **by** *metis*  
**next**  
**qed**

**lemma** *parallel-reduces-diamond*:  
**assumes**  $A \gg^* B \ A \gg^* C$   
**shows**  $\exists D. (B \gg^* D) \wedge (C \gg^* D)$   
**using** *assms* **proof**(*induction*)  
**case** (*reflexive*  $A$ )  
**thus** *?case* **using** *parallel-reduces.reflexive* **by** *metis*  
**next**  
**case** (*transitive*  $A \ A' \ A''$ )  
**from** *this* **obtain**  $C'$  **where**  $A' \gg^* C' \ C \gg^* C'$  **by** *metis*  
**from** *this* **obtain**  $D$  **where**  $A'' \gg^* D \ C' \gg D$   
**using**  $\langle A' \gg A'' \rangle \langle A' \gg^* C' \rangle$  *parallel-reduces-diamond'* **by** *metis*  
**thus** *?case* **using** *parallel-reduces.transitive*  $\langle C \gg^* C' \rangle$  **by** *metis*  
**next**  
**qed**

**lemma** *beta-reduction-is-parallel-reduction*:  
**assumes**  $A \rightarrow \beta B$   
**shows**  $A \gg B$   
**using** *assms*



```

apply induction
apply (metis parallel-reduction.beta parallel-reduction.refl)
apply (metis parallel-reduction.app parallel-reduction.refl)
apply (metis parallel-reduction.app parallel-reduction.refl)
apply (metis parallel-reduction.eta)
apply (metis parallel-reduction.pair parallel-reduction.refl)
apply (metis parallel-reduction.pair parallel-reduction.refl)
apply (metis parallel-reduction.fst1)
apply (metis parallel-reduction.fst2 parallel-reduction.refl)
apply (metis parallel-reduction.snd1)
apply (metis parallel-reduction.snd2 parallel-reduction.refl)
done

```

**lemma** *parallel-reduction-is-beta-reduction*:

```

assumes  $A \gg B$ 
shows  $A \rightarrow\beta^* B$ 
using assms proof(induction)
case (refl A)
  thus ?case using beta-reduces.reflexive.
next
case (beta A A' B B' x T)
  hence  $(App (Fn x T A) B) \rightarrow\beta^* (App (Fn x T A') B')$ 
  using  $\langle A \rightarrow\beta^* A' \rangle$ 
  beta-reduces-fn beta-reduces-app-left beta-reduces-app-right beta-reduces-composition
  by metis
  moreover have  $(App (Fn x T A') B') \rightarrow\beta (A'[x ::= B'])$ 
  using beta-reduction.beta.
  ultimately show ?case using beta-reduces.transitive by metis
next
case (eta A A' x T)
  thus ?case using beta-reduces-fn by metis
next
case (app A A' B B')
  thus ?case using beta-reduces-app-left beta-reduces-app-right beta-reduces-composition
by metis
next
case (pair A A' B B')
  thus ?case using beta-reduces-pair-left beta-reduces-pair-right beta-reduces-composition
by metis
next
case (fst1 P P')
  thus ?case using beta-reduces-fst by metis
next
case (fst2 A A' B)
  thus ?case
  using beta-reduces-pair-left beta-reduction.fst2 beta-reduces.intros beta-reduces-composition
  by blast
next
case (snd1 P P')

```

```

    thus ?case using beta-reduces-snd by metis
  next
  case (snd2 B B' A)
    thus ?case
    using beta-reduces-pair-left beta-reduction.snd2 beta-reduces.intros beta-reduces-composition
    by blast
  next
  qed

```

```

lemma parallel-beta-reduces-equivalent:
  shows  $(A \gg^* B) = (A \rightarrow\beta^* B)$ 
  proof -
    have  $\rightarrow: (A \gg^* B) \implies (A \rightarrow\beta^* B)$ 
    proof(induction rule: parallel-reduces.induct)
      case (reflexive A)
        thus ?case using beta-reduces.reflexive.
      next
      case (transitive A A' A'')
        thus ?case using beta-reduces-composition parallel-reduction-is-beta-reduction
    by metis
    next
    qed

```

```

  have  $\leftarrow: (A \rightarrow\beta^* B) \implies (A \gg^* B)$ 
  proof(induction rule: beta-reduces.induct)
    case (reflexive A)
      thus ?case using parallel-reduces.reflexive.
    next
    case (transitive A A' A'')
      thus ?case using parallel-reduces.transitive beta-reduction-is-parallel-reduction
  by metis
  next
  qed

```

```

  from  $\leftarrow \rightarrow$  show  $(A \gg^* B) = (A \rightarrow\beta^* B)$  by blast
  qed

```

```

theorem confluence:
  assumes  $A \rightarrow\beta^* B$   $A \rightarrow\beta^* C$ 
  shows  $\exists D. (B \rightarrow\beta^* D) \wedge (C \rightarrow\beta^* D)$ 
  proof -
    from assms have  $A \gg^* B$   $A \gg^* C$  using parallel-beta-reduces-equivalent by
  metis+
    hence  $\exists D. (B \gg^* D) \wedge (C \gg^* D)$  using parallel-reduces-diamond by metis
    thus  $\exists D. (B \rightarrow\beta^* D) \wedge (C \rightarrow\beta^* D)$  using parallel-beta-reduces-equivalent by
  metis
  qed

```

```

end

```

**end**