Verified Metatheory and Type Inference for a Name-Carrying Simply-Typed \( \lambda \)-Calculus

Michael Rawson

April 20, 2020

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 \Rightarrow 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 bymetis
    hence 0 \notin S by auto
    thus (if Set.is-empty S then 0 else Max S + 1) \notin S using I by auto
next
case 2
have $\text{Max } S + 1 \notin S$
    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
    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
    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 p = 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 ≠ y
    shows swp-apply s x ≠ swp-apply s y
proof(cases s)
    case (Pair a b)
    consider x = a | x = b | x ≠ a ∧ x ≠ 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 ≠ b using (s = (a, b)) (x = a) (x ≠ y)
by\langle \text{cases } y = b, \text{ simp-all} \rangle
\text{ultimately show } ?\text{thesis by metis}
next
case 2
have \text{swp-apply } s \times = a \text{ using } \langle s = (a, b) \rangle \langle x = b \rangle \text{ by simp}
moreover have \text{swp-apply } s \times y \neq a \text{ using } \langle s = (a, b) \rangle \langle x = b \rangle \langle x \neq y \rangle
\text{by}(\text{cases } y = a, \text{ simp-all})
\text{ultimately show } ?\text{thesis by metis}
next
case 3
have \text{swp-apply } s \times = x \text{ using } \langle s = (a, b) \rangle \langle x \neq a \text{ and } x \neq b \rangle \text{ by simp}
consider \begin{align*}
y &= a \mid y &= b \mid y &\neq a \text{ and } y \neq b \end{align*} \text{ by auto}
hence \text{swp-apply } s \times y \neq x \text{ proof}(\text{cases})
\begin{align*}
case 1 & \text{ hence swp-apply } s \times y = b \text{ using } \langle s = (a, b) \rangle \text{ by simp}
& \text{ thus } ?\text{thesis using } \langle x \neq a \text{ and } x \neq b \rangle \text{ by metis}
\end{align*}
next
case 2
\begin{align*}
& \text{ hence swp-apply } s \times y = a \text{ using } \langle s = (a, b) \rangle \text{ by simp}
& \text{ thus } ?\text{thesis using } \langle x \neq a \text{ and } x \neq b \rangle \text{ by metis}
\end{align*}
next
case 3
\begin{align*}
& \text{ hence swp-apply } s \times y = y \text{ using } \langle s = (a, b) \rangle \text{ by simp}
& \text{ thus } ?\text{thesis using } \langle x \neq y \rangle \text{ by metis}
\end{align*}
next
qed
\begin{align*}
& \text{ thus } ?\text{thesis using } \langle \text{swp-apply } s \times = x \rangle \langle x \neq y \rangle \text{ by metis}
\end{align*}
next
qed

\text{lemma preprm-ext-reflexive:}
\begin{align*}
& \text{shows } x \rightarrow p x
\end{align*}
\text{unfolding preprm-ext-def by auto}

\text{corollary preprm-ext-reflp:}
\begin{align*}
& \text{shows reflp preprm-ext}
\end{align*}
\text{unfolding reflp-def using preprm-ext-reflexive by auto}

\text{lemma preprm-ext-symmetric:}
\begin{align*}
& \text{assumes } x \rightarrow p y
& \text{shows } y \rightarrow p x
\end{align*}
\text{using assms unfolding preprm-ext-def by auto}

\text{corollary preprm-ext-symp:}
\begin{align*}
& \text{shows symp preprm-ext}
\end{align*}
\text{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 \((s \# ss) x = swallop s (preprm-apply ss x)\)
  and preprm-apply \((s \# ss) y = swallop s (preprm-apply ss y)\) by auto
  thus \( \text{case using Cons.IH \( \langle x \neq y \rangle \) swallop-unequal by metis} \)next
qed

lemma preprm-unit-equal-id:
  shows preprm-unit a a = p 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 = p preprm-unit b a
unfolding preprm-ext-def preprm-unit-def
by simp

lemma preprm-singleton-involution:
  shows preprm-compose [s] [s] = p preprm-id
unfolding preprm-ext-def preprm-compose-def preprm-unit-def preprm-id-def
proof –
obtain \( s_1 \ s_2 \) where \( s_1 = \text{fst} \ s \ s_2 = \text{snd} \ s \) by auto

hence \( s = (s_1, \ s_2) \) by simp

thus \( \forall \ x. \ \text{preprm-apply} \ ([s] \ @ \ [s]) \ x = \text{preprm-apply} \ [\ ] \ x \)

by simp

qed

lemma \text{preprm-unit-involution}:

shows \( \text{preprm-compose} \ (\text{preprm-unit} \ a \ b) \ (\text{preprm-unit} \ a \ b) = \text{preprm-id} \)

unfolding \text{preprm-compose-def} \ using \text{preprm-singleton-involution}.

lemma \text{preprm-apply-id}:

shows \( \text{preprm-apply} \ \text{preprm-id} \ x = x \)

unfolding \text{preprm-id-def} \ by simp

lemma \text{preprm-apply-injective}:

shows \( \text{inj} \ (\text{preprm-apply} \ \pi) \)

unfolding \text{inj-on-def} \ proof(rule+)

fix \( x \ y \)

assume \( \text{preprm-apply} \ \pi \ x = \text{preprm-apply} \ \pi \ y \)

thus \( x = y \) proof(induction \( \pi \))

case Nil

thus \( ?\text{case} \) by auto

next

case (Cons \( s \ ss \))

hence \( \text{swp-apply} \ s \ (\text{preprm-apply} \ ss \ x) = \text{swp-apply} \ s \ (\text{preprm-apply} \ ss \ y) \) by auto

thus \( ?\text{case} \) using \( \text{swp-apply-unequal} \ \text{Cons.IH} \) by metis

next

qed

qed

lemma \text{preprm-disagreement-composition}:

assumes \( a \neq b \ b \neq c \ a \neq c \)

shows \( \{x. \ \text{preprm-apply} \ (\text{preprm-compose} \ (\text{preprm-unit} \ a \ b) \ (\text{preprm-unit} \ b \ c)) \ x \neq \text{preprm-apply} \ (\text{preprm-unit} \ a \ c) \ x\} = \{a, \ b\} \)

unfolding \text{preprm-unit-def} \ \text{preprm-compose-def} \ proof

show \( \{x. \ \text{preprm-apply} \ ([\ (a, \ b)\ ] \ @ \ [\ (b, \ c)\ ]) \ x \neq \text{preprm-apply} \ [(a, \ c)] \ x\} \subseteq \{a, \ b\} \)

proof

fix \( x \)

have \( x \not\in \{a, \ b\} \implies x \not\in \{x. \ \text{preprm-apply} \ ([\ (a, \ b)\ ] \ @ \ [\ (b, \ c)\ ]) \ x \neq \text{preprm-apply} \ [(a, \ c)] \ x\} \)

proof –

assume \( x \not\in \{a, \ b\} \)

hence \( x \neq a \land x \neq b \) by auto
hence \( \text{preprm-apply} \left( \left[ \left( a, b \right) \right] @ \left[ \left( b, c \right) \right] \right) \) \( x = \text{preprm-apply} \left( \left[ \left( a, c \right) \right] \right) \) by simp

thus \( x \notin \left\{ x. \text{preprm-apply} \left( \left[ \left( a, b \right) \right] @ \left[ \left( b, c \right) \right] \right) \neq \text{preprm-apply} \left( \left[ \left( a, c \right) \right] \right) \right\} \) by auto

qed

thus \( x \in \left\{ a, b \right\} \) by blast

qed

show \( \{ a, b \} \subseteq \left\{ x. \text{preprm-apply} \left( \left[ \left( a, b \right) \right] @ \left[ \left( b, c \right) \right] \right) \neq \text{preprm-apply} \left( \left[ \left( a, c \right) \right] \right) \right\} \)

proof

fix \( x \)

assume \( x \in \{ a, b \} \)

from this consider \( x = a \mid x = b \) by auto

thus \( x \in \left\{ x. \text{preprm-apply} \left( \left[ \left( a, b \right) \right] @ \left[ \left( b, c \right) \right] \right) \neq \text{preprm-apply} \left( \left[ \left( a, c \right) \right] \right) \right\} \)

using assms by (cases, simp-all)

qed

qed

lemma \( \text{preprm-compose-push} \):

shows

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

unfolding \( \text{preprm-ext-def} \, \text{preprm-unit-def} \)

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

lemma \( \text{preprm-ext-compose-left} \):

assumes \( P =p S \)

shows \( \text{preprm-compose} \, \pi \, P =p \, \text{preprm-compose} \, \pi \, S \)

using assms unfolding \( \text{preprm-ext-def} \)

using \( \text{preprm-apply-composition} \) by metis

lemma \( \text{preprm-ext-compose-right} \):

assumes \( P =p S \)

shows \( \text{preprm-compose} \, P \, \pi =p \, \text{preprm-compose} \, S \, \pi \)

using assms unfolding \( \text{preprm-ext-def} \)

using \( \text{preprm-apply-composition} \) by metis

lemma \( \text{preprm-ext-uncompose} \):

assumes \( \pi =p \, \sigma \, \text{preprm-compose} \, \pi \, P =p \, \text{preprm-compose} \, \sigma \, S \)

shows \( P =p \, S \)

using assms unfolding \( \text{preprm-ext-def} \)

proof

assume \( \ast : \forall \, x. \text{preprm-apply} \, \pi \, x = \text{preprm-apply} \, \sigma \, x \)

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

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

using \( \text{preprm-apply-composition} \) by \( \text{metis} \) 

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

using \( \ast \) by \( \text{metis} \) 

thus \( \forall x. \text{preprm-apply } P \ x = \text{preprm-apply } S \ x \) 

using \( \text{preprm-apply-injective} \) unfolding \( \text{inj-on-def} \) by \( \text{fastforce} \)

\( \text{qed} \)

\text{lemma} \( \text{preprm-inv-compose} \): 
shows \( \text{preprm-compose} \ (\text{preprm-inv} \ \pi) \ \pi = \text{p preprm-id} \)
unfolding \( \text{preprm-compose-def} \)
proof (induction \( \pi \), simp add: \( \text{preprm-ext-def} \ \text{preprm-id-def} \ \text{preprm-compose-def} \))
  case \( \text{Cons} \ p \ ps \)
  hence \( \text{IH}: (\text{preprm-compose} \ (\text{rev} \ ps) \ ps) = \text{p preprm-id by auto} \)

  have \( \text{(preprm-compose} \ (\text{rev} \ (p \ # \ ps)) \ (p \ # \ ps)) = \text{p (preprm-compose} \ (\text{rev} \ ps) \ (\text{preprm-compose} \ [p] \ [p]) \ ps)) \)
unfolding \( \text{preprm-compose-def} \) using \( \text{preprm-ext-reflexive} \) by \( \text{simp} \)

  moreover have \( \ldots = \text{p (preprm-compose} \ (\text{rev} \ ps) \ (\text{preprm-compose} \ \text{preprm-id} \ ps)))) \)
using \( \text{preprm-singleton-involution} \ \text{preprm-ext-compose-left} \ \text{preprm-ext-compose-right} \)
by \( \text{metis} \)

  moreover have \( \ldots = \text{p preprm-id using IH} \).

ultimately show ?case using \( \text{preprm-ext-transitive by metis} \)
next
\( \text{qed} \)

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

\text{lemma} \( \text{preprm-inv-ext} \): 
assumes \( \pi = \text{p } \sigma \)
shows \( \text{preprm-inv} \ \pi = \text{p preprm-inv } \sigma \)
proof –

  have \( \text{(preprm-compose} \ (\text{preprm-inv} \ (\text{preprm-inv} \ \pi)) \ (\text{preprm-inv} \ \pi)) = \text{p preprm-id} \)
\( \text{(preprm-compose} \ (\text{preprm-inv} \ (\text{preprm-inv} \ \sigma)) \ (\text{preprm-inv} \ \sigma)) = \text{p preprm-id} \)
using \( \text{preprm-inv-compose by metis+} \)

hence \( \text{(preprm-compose} \ \pi \ (\text{preprm-inv} \ \pi)) = \text{p preprm-id} \)
\( \text{(preprm-compose} \ \sigma \ (\text{preprm-inv} \ \sigma)) = \text{p preprm-id} \)
using \( \text{preprm-inv-involution by metis+} \)

hence \( \text{(preprm-compose} \ \pi \ (\text{preprm-inv} \ \pi)) = \text{p (preprm-compose} \ \sigma \ (\text{preprm-inv} \ \sigma)) \)

7
using preprm-ext-transitive preprm-ext-symmetric by metis
thus preprm-inv π = p preprm-inv σ
using preprm-ext-uncompose assms by metis
qed

quotient-type 'a prm = 'a preprm / preprm-ext
proof(rule equivpl)
  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 (ε) is preprm-id.

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

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

lift-definition prm-unit :: 'a ⇒ 'a prm ([− ↔ −]) is preprm-unit.

lift-definition prm-inv :: 'a prm ⇒ 'a prm is preprm-inv
using preprm-inv-ext.

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

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

lemma prm-unit-equal-id:
  fixes a :: 'a
  shows [a ↔ a] = ε
by (transfer, metis preprm-unit-equal-id)

lemma prm-unit-inaction:
  fixes a b x :: 'a
  assumes x ≠ a and x ≠ b
  shows [a ↔ b] $ x = x
using assms
by (transfer, metis preprm-unit-inaction)

lemma prm-unit-action:
  fixes a b :: 'a
  shows [a ↔ b] $ a = b
by (transfer, metis preprm-unit-action)

lemma prm-unit-commutes:
  fixes a b :: 'a
  shows [a ↔ b] = [b ↔ a]
by (transfer, metis preprm-unit-commutes)

lemma prm-unit-involution:
  fixes a b :: 'a
  shows [a ↔ b] ◦ [a ↔ b] = ε
by (transfer, metis preprm-unit-involution)

lemma prm-apply-id:
  fixes x :: 'a
  shows ε $ x = x
by (transfer, metis preprm-apply-id)

lemma prm-apply-injective:
  shows inj (prm-apply π)
by (transfer, metis preprm-apply-injective)

lemma prm-inv-compose:
  shows (prm-inv π) ◦ π = ε
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 (◦) using 'a prm.semidgroup-axioms.
  moreover have ∀ a. ε ◦ a = a by (transfer, simp add: preprm-id-def preprm-compose-def
  preprm-ext-def)
  moreover have ∀ a. prm-inv a ◦ a = ε using prm-inv-compose by blast
  ultimately show semigroup (◦) ∧ (∀ a. ε ◦ a = a) ∧ (∀ a. prm-inv a ◦ a = ε)
  by blast
qed

definition prm-set :: 'a prm ⇒ 'a set ⇒ 'a set (infix {§} 140) where
  prm-set π S ≡ image (prm-apply π) S

lemma prm-set-compose:
  shows π {§} (σ {§} S) = (π ◦ σ) {§} S
unfolding \textit{prm-set-def} \textbf{proof} –
have \((\varepsilon) \pi \cdot (\varepsilon) \sigma \cdot S = (\lambda x. \pi \cdot x) \cdot (\lambda x. \sigma \cdot x) \cdot S\) by \textit{simp}
moreover have \(\ldots = (\lambda x. \pi \cdot (\sigma \cdot x)) \cdot S\) by \textit{auto}
moreover have \(\ldots = (\lambda x. (\pi \circ \sigma) \cdot x) \cdot S\) using \textit{prm-apply-composition by metis}
moreover have \(\ldots = (\pi \circ \sigma) \cdot S\) by \textit{auto}
ultimately show \((\varepsilon) \pi \cdot (\varepsilon) \sigma \cdot S = (\varepsilon) \cdot (\pi \circ \sigma) \cdot S\) by \textit{metis}
\textbf{qed}

\textbf{lemma} \textit{prm-set-membership}:
\textbf{assumes} \(x \in S\)
\textbf{shows} \(\pi \cdot x \in \pi \cdot \{x\} \cdot S\)
\textbf{using} \textit{assms unfolding} \textit{prm-set-def by simp}

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

\textbf{lemma} \textit{prm-set-singleton}:
\textbf{shows} \(\pi \cdot \{x\} \cdot \{x\} = \{\pi \cdot x\}\)
\textbf{unfolding} \textit{prm-set-def by auto}

\textbf{lemma} \textit{prm-set-id}:
\textbf{shows} \(\varepsilon \cdot \{x\} \cdot S = S\)
\textbf{unfolding} \textit{prm-set-def}
\textbf{proof} –
\textbf{have} \((\varepsilon) \in \cdot S = (\lambda x. \varepsilon \cdot x) \cdot S\) by \textit{simp}
\textbf{moreover have} \(\ldots = (\lambda x. x) \cdot S\) using \textit{prm-apply-id by metis}
\textbf{moreover have} \(\ldots = S\) by \textit{auto}
\textbf{ultimately show} \((\varepsilon) \cdot S = S\) by \textit{metis}
\textbf{qed}

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

\textbf{lemma} \textit{prm-set-unit-action}:
- assumes $a \in S$ and $b \notin S$
- shows $[a \leftrightarrow b] \ \{\$\} \ S = S - \{a\} \cup \{b\}$
- \textbf{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 \textit{prm-set-def} using \textit{imageE} by \textit{metis}
    - hence $y \in S$ using $H$ \textit{inj-image-mem-iff} \textit{prm-apply-injective} \textit{prm-set-def} by \textit{metis}
    - hence $y \neq b$ using \textit{assms} by \textit{auto}
    - consider $y = a$ | $y \neq a$ by \textit{auto}
    - thus $x \in S$ proof \textit{(cases)}
      - case 1
        - hence $x = b$ using $x = [a \leftrightarrow b] \ y$ using \textit{prm-unit-action} by \textit{metis}
        - thus \textit{thesis} by \textit{auto}
      - next
      - case 2
        - hence $x = y$ using $x = [a \leftrightarrow b] \ y$ using \textit{prm-unit-inaction} \textit{y \neq b} by \textit{metis}
        - hence $x \in S$ and $x \neq a$ using $y \in S$. \textit{y \neq a} by \textit{auto}
        - thus \textit{thesis} by \textit{auto}
      - next
    - qed
  - \textbf{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 \textit{assms} by \textit{auto}
  - consider $x = b$ | $x \neq b$ by \textit{auto}
  - thus $x \in [a \leftrightarrow b] \ \{\$\} \ S$ proof \textit{(cases)}
    - case 1
      - hence $x = [a \leftrightarrow b] \ a$ using \textit{prm-unit-action} by \textit{metis}
      - thus \textit{thesis} using $a \in S$. \textit{prm-set-membership} by \textit{metis}
    - next
    - case 2
      - hence $x \in S$ using $H$ by \textit{auto}
      - moreover have $x = [a \leftrightarrow b] \ x$ using \textit{prm-unit-inaction} \textit{x \neq a}. \textit{x \neq b}
        by \textit{metis}
ultimately show \( \text{thesis using} \) \( \text{prm-set-membership by} \) \( \text{metis} \)
next
qed
qed
qed

\textbf{lemma} \( \text{prm-set-distributes-union:} \)
\begin{align*}
&\text{shows } \pi \{x \in S \cup T \mid \pi x \neq \sigma x\} = (\pi \{x \in S \mid \pi x \neq \sigma x\}) \cup (\pi \{x \in T \mid \pi x \neq \sigma x\}) \\
&\text{unfolding} \quad \text{prm-set-def by auto}
\end{align*}

\textbf{lemma} \( \text{prm-set-distributes-difference:} \)
\begin{align*}
&\text{shows } \pi \{x \in S \setminus T \mid \pi x \neq \sigma x\} = (\pi \{x \in S \mid \pi x \neq \sigma x\}) \setminus (\pi \{x \in T \mid \pi x \neq \sigma x\}) \\
&\text{unfolding} \quad \text{prm-set-def using} \quad \text{prm-apply-injective image-set-diff by metis}
\end{align*}

\textbf{definition} \( \text{prm-disagreement :: 'a ptrm ⇒ 'a ptrm ⇒ 'a set (ds) where} \)
\begin{align*}
&\text{prm-disagreement} \pi \sigma \equiv \{x . \pi x \neq \sigma x\} \\
&\text{unfolding} \quad \text{prm-disagreement-def by simp}
\end{align*}

\textbf{lemma} \( \text{prm-disagreement-composition:} \)
\begin{align*}
&\text{assumes} \quad a \neq b \quad b \neq c \quad a \neq c \\
&\text{shows} \quad ds (([a ↔ b] \circ [b ↔ c]) \circ [a ↔ c]) = \{a, b\} \\
&\text{using} \quad \text{assms unfolding} \quad \text{prm-disagreement-def by(transfer, metis preprm-disagreement-composition)}
\end{align*}

\textbf{lemma} \( \text{prm-compose-push:} \)
\begin{align*}
&\text{shows} \quad \pi \circ ([a ↔ b]) = [\pi a ↔ \pi b] \circ \pi \\
&\text{by(transfer, metis preprm-compose-push)}
\end{align*}

end

\textbf{theory} \( \text{PreSimplyTyped} \)
\textbf{imports} \( \text{Fresh Permutation} \)
\textbf{begin}

\textbf{type-synonym} \( \text{tvar} = \text{nat} \)

\textbf{datatype} \( \text{type} = \)
\begin{align*}
&TUnit \\
|&TVar tvar \\
|&TArr type type \\
|&TPair type type
\end{align*}

\textbf{datatype} \( \text{'a ptrm} = \)
\begin{align*}
&PUnit \\
|&PVar 'a \\
|&PApp 'a ptrm 'a ptrm \\
|&PFN 'a type 'a ptrm \\
|&PPair 'a ptrm 'a ptrm
\end{align*}
\begin{verbatim}
| PFst 'a ptrm
| PSnd 'a ptrm

fun ptrm-fvs :: 'a ptrm ⇒ 'a set where
  ptrm-fvs PUnit = {}
  ptrm-fvs (PVar x) = {x}
  ptrm-fvs (PApp A B) = ptrm-fvs A ∪ ptrm-fvs B
  ptrm-fvs (PFN x - A) = ptrm-fvs A - {x}
  ptrm-fvs (PPair A B) = ptrm-fvs A ∪ ptrm-fvs B
  ptrm-fvs (PFst P) = ptrm-fvs P
  ptrm-fvs (PSnd P) = ptrm-fvs P

fun ptrm-apply-prm :: 'a prm ⇒ 'a ptrm ⇒ 'a ptrm (infixr · 150) where
  ptrm-apply-prm π PUnit = PUnit
  ptrm-apply-prm π (PVar x) = PVar (π $ x)
  ptrm-apply-prm π (PApp A B) = PApp (ptrm-apply-prm π A) (ptrm-apply-prm π B)
  ptrm-apply-prm π (PFN x T A) = PFn (π $ x) T (ptrm-apply-prm π A)
  ptrm-apply-prm π (PPair A B) = PPair (ptrm-apply-prm π A) (ptrm-apply-prm π B)
  ptrm-apply-prm π (PFst P) = PFst (ptrm-apply-prm π P)
  ptrm-apply-prm π (PSnd P) = PSnd (ptrm-apply-prm π P)

inductive ptrm-alpha-equiv :: 'a ptrm ⇒ 'a ptrm ⇒ bool (infix ≈ 100) where
  unit:  PUnit ≈ PUnit
  var:  (PVar x) ≈ (PVar x)
  app:  [A ≈ B; C ≈ D] ⇒ (PApp A C) ≈ (PApp B D)
  fn1:  A ≈ B ⇒ (PFN x T A) ≈ (PFN x T B)
  fn2:  A ≈ B ⇒ (PFN a T A) ≈ (PFN a T B)
  pair: [A ≈ B; C ≈ D] ⇒ (PPair A C) ≈ (PPair B D)
  fst: A ≈ B ⇒ PFst A ≈ PFst B
  snd: A ≈ B ⇒ PSnd A ≈ PSnd B

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

lemma ptrm-prm-apply-id:
  shows ε · X = X
  by(induction X, simp-all add: prm-apply-id)

lemma ptrm-prm-apply-compose:
  shows π · σ · X = (π o σ) · X
  by(induction X, simp-all add: prm-apply-composition)
\end{verbatim}
lemma ptrm-size-prm:
  shows size $X = size (\pi \cdot X)$
by(induction $X$, auto)

lemma ptrm-size-alpha-equiv:
  assumes $X \approx Y$
  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 ($\pi \cdot X$) = $\pi \{\}$ ptrm-fvs $X$
proof(induction $X$)
  case (PUnit)
    thus $?case$ unfolding prm-set-def by simp
  next
  case (PVar $x$
    have ptrm-fvs ($\pi \cdot PVar \; x$) = ptrm-fvs ($PVar \; (\pi \; \{\} \; x)$) by simp
    moreover have ... = $\pi \{\} \; x$ by simp
    moreover have ... = $\pi \{\} \; \{x\}$ using prm-set-singleton by metis
    moreover have ... = $\pi \{\} \; ptrm-fvs \; (PVar \; x)$ by simp
    ultimately show $?case$ by metis
  next
  case (PApp $A \; B$
    have ptrm-fvs ($\pi \cdot PApp \; A \; B$) = ptrm-fvs ($PApp \; (\pi \cdot A) \cdot (\pi \cdot B)$) by simp
    moreover have ... = ptrm-fvs ($\pi \cdot A$) $\cup$ ptrm-fvs ($\pi \cdot B$) by simp
    moreover have ... = $\pi \{\} \; ptrm-fvs \; A \cup \pi \{\} \; ptrm-fvs \; B$ using PApp.IH
    by metis
    moreover have ... = $\pi \{\} \; (ptrm-fvs \; A \cup ptrm-fvs \; B)$ using prm-set-distributes-union
    by metis
    moreover have ... = $\pi \{\}$ ptrm-fvs ($PApp \; A \; B$) by simp
    ultimately show $?case$ by metis
  next
  case (PFn $x \; T \; A$
    have ptrm-fvs ($\pi \cdot PFn \; x \; T \; A$) = ptrm-fvs ($PFN \; (\pi \; \{\} \; x) \; T \cdot (\pi \cdot A)$) by simp
    moreover have ... = ptrm-fvs ($\pi \cdot A$) $-$ $\pi \{\} \; x$ by simp
    moreover have ... = $\pi \{\} \; ptrm-fvs \; A \; - \; \pi \{\} \; \{x\}$ using PFn.IH by metis
    moreover have ... = $\pi \{\} \; ptrm-fvs \; A \; - \; \pi \{\} \; \{x\}$ using prm-set-singleton
    by metis
    moreover have ... = $\pi \{\} \; (ptrm-fvs \; A \; - \; \{x\})$ using prm-set-distributes-difference
  next
  qed
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 ptrm-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 ptrm-fvs B | b \notin 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 ptrm-unit-commutes by metis
    moreover have ... = ((ptrm-fvs B - \{b\}) \cup \{a\}) - \{a\}
      using ptrm-set-unit-action \b \in ptrm-fvs B \{a \notin ptrm-fvs B\} by metis
    moreover have ... = ptrm-fvs B - \{b\} using \a \notin ptrm-fvs B \a \neq b
      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 ptrm-set-unit-inaction \a \notin ptrm-fvs B by metis
  moreover have ... = ptrm-fvs B using \a \notin ptrm-fvs B by simp
  moreover have ... = ptrm-fvs B - \{b\} using \b \notin ptrm-fvs B by simp
  ultimately show \?thesis by metis
next
qed
qed
moreover have ... = ptrm-fvs (PFn b T B) by simp
ultimately show ?case by metis
next
qed auto

lemma ptrm-alpha-eqiv-prm:
assumes X ≈ Y
shows π · X ≈ π · 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 π $ a ≠ π $ b using ⟨a ≠ b⟩ using prm-apply-unequal by metis
    moreover have π $ a /∈ ptrm-fvs (π · B) using ⟨a /∈ ptrm-fvs B⟩ using imageE prm-apply-unequal prm-set-def ptrm-prm-fvs by (metis (no-types, lifting))
    moreover have π · A ≈ [π $ a ↔ π $ b] · π · B
      using fn2.IH prm-compose-push ptrm-prm-compose-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 ↔ b] · X ≈ Y ↔ X ≈ [a ↔ b] · Y
proof
  have 1: [a ↔ b] · X ≈ Y ⊢ X ≈ [a ↔ b] · Y
  proof
    assume [a ↔ b] · X ≈ Y
    hence ε · X ≈ [a ↔ b] · Y
      using ptrm-alpha-equiv-prm ptrm-prm-compose-compose ptrm-unit-involution by

metis

thus ?thesis using ptrm-prm-apply-id by metis
qed

have 2: X \approx [a \leftrightarrow b] \cdot Y \Rightarrow [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 \leftrightarrow 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 ptrm-fvs B
  shows b \notin 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 ptrm-fvs B = [a \leftrightarrow b] \{\} ptrm-fvs A
  using ptrm-alpha-equiv-fvs ptrm-prm-fvs by metis
  hence a \notin [a \leftrightarrow b] \{\} ptrm-fvs A using \langle a \notin ptrm-fvs B \rangle by metis
  hence b \notin [a \leftrightarrow b] \{\} ([a \leftrightarrow b] \{\} ptrm-fvs A)
  using prm-set-notmembership prm-unit-action by metis
thus ?thesis using prm-set-apply-compose prm-unit-involution prm-set-id by metis
qed

lemma ptrm-prm-agreement-equiv:
  assumes \forall a. a \in ds \pi \sigma \Rightarrow a \notin 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 ds \pi \sigma | x \notin ds \pi \sigma by auto
  thus ?case proof (cases)
  case 1
    hence x \notin ptrm-fvs (PVar x) using PVar.prems by blast
    thus ?thesis by simp
  next
case 2
  hence \pi \$ x = \sigma \$ x using prm-disagreement-ext by metis
  thus ?thesis using ptrm-alpha-equiv.var by simp
next
qed
next
\textbf{case} (PApp A B)

hence $\pi \cdot A \approx \sigma \cdot A$ and $\pi \cdot B \approx \sigma \cdot B$ by auto

thus $\text{	extbf{case using ptrm-alpha-equiv.app by auto}}$

\textbf{next}

\textbf{case} (PFN x T A)

consider $x \notin ds \pi \sigma \mid x \in ds \pi \sigma$ by auto

thus $\text{	extbf{case proof(cases)}}$

\textbf{case 1}

hence $*: \pi \{ x \} = \sigma \{ x \}$ using \text{prm-disagreement-ext} by metis

have $\bigwedge \{ a. \ a \in ds \pi \sigma \Rightarrow a \notin ptrm-fes A \}$

proof –

fix a

assume $a \in ds \pi \sigma$

hence $a \notin ptrm-fes (PFN x T A)$ using PFn.prems by metis

hence $a = x \lor a \notin ptrm-fes A$ by auto

thus $a \notin ptrm-fes A$ using $(a \in ds \pi \sigma \mid x \notin ds \pi \sigma)$ by auto

qed

thus $\text{?thesis using PFn.IH * ptrm-alpha-equiv.fn1 ptrm-apply-prm.simps(3)}$

by \text{fastforce}

\textbf{next}

\textbf{case 2}

hence $\pi \{ x \} = \sigma \{ x \}$ using \text{prm-disagreement-def CollectD} by \text{fastforce}

moreover have $\pi \{ x \} \notin ptrm-fes (\sigma \cdot A)$

proof –

have $y \in (ptrm-fes A) \Rightarrow \pi \{ x \} = \sigma \{ y \}$ for y

using PFn.($\pi \{ x \} \neq \sigma \{ x \}$) \text{ptrm-fvs A} unfolding \text{prm-def} by auto

by \text{(metis Diff-iff empty-iff insert-iff)}

hence $\pi \{ x \} \notin \sigma \{ \}$ \text{ptrm-fes A} \text{unfolding} \text{prm-def} by auto

thus $\text{?thesis using ptrm-fv A by metis}$

qed

moreover have $\pi \cdot A \approx [\pi \{ x \} \leftrightarrow \sigma \{ x \}] \cdot \sigma \cdot A$

proof –

have $\bigwedge \{ a. \ a \in ds \pi (\{ \pi \{ x \} \leftrightarrow \sigma \{ x \} \circ \sigma \}) \Rightarrow a \notin ptrm-fes A \}$

proof –

fix a

assume $*: a \in ds \pi (\{ \pi \{ x \} \leftrightarrow \sigma \{ x \} \circ \sigma \})$

hence $a \neq x$ using $(\pi \{ x \} \neq \sigma \{ x \})$

using \text{ptrm-apply-composition ptrm-disagreement-ext ptrm-unit-action ptrm-unit-commutes ptrm-unit-inaction}

by metis

hence $a \in ds \pi \sigma$

using $* \text{ ptrm-apply-composition ptrm-apply-unequal ptrm-disagreement-ext ptrm-unit-inaction}$

by metis

thus $a \notin ptrm-fes A$ using $\{ a \neq x \}$ PFn.prems by auto

qed

thus $\text{?thesis using PFn by (simp add: ptrm-apply-compose)}$

qed

ultimately show $\text{?thesis using ptrm-alpha-equiv.fn2 by simp}$
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 \( ?\text{case using ptrm-alpha-equiv.pair by auto} \)
next
case (PFst P)
  hence \( \pi \cdot P \approx \sigma \cdot P \) by auto
  thus \( ?\text{case using ptrm-alpha-equiv.fst by auto} \)
next
case (PSnd P)
  hence \( \pi \cdot P \approx \sigma \cdot P \) by auto
  thus \( ?\text{case using ptrm-alpha-equiv snd by auto} \)
next
qed

lemma ptrm-prm-unit-inaction:
  assumes \( a \not\in \text{ptrm-fvs X} \) \( b \not\in \text{ptrm-fvs X} \)
  shows \( [a \leftrightarrow b] \cdot X \approx X \)
proof -
  have \( (\forall x. x \in ds \ [a \leftrightarrow b] \varepsilon \implies x \notin \text{ptrm-fvs X}) \)
proof -
  fix \( x \)
  assume \( x \in ds \ [a \leftrightarrow b] \varepsilon \)
  hence \( [a \leftrightarrow b] \# x \notin \varepsilon \$ x \)
  unfolding \( \text{prm-disagreement-def} \)
  by auto
  hence \( x = a \lor x = b \)
  using \( \text{prm-apply-id} \) \( \text{ptrm-alpha-equiv} \) by \( \text{metis} \)
  thus \( x \notin \text{ptrm-fvs X} \) using \( \text{assms by auto} \)
  qed
  hence \( [a \leftrightarrow b] \cdot X \approx \varepsilon \cdot X \)
  using \( \text{ptrm-prm-agreement-equiv} \) by \( \text{metis} \)
  thus \( ?\text{thesis using ptrm-prm-apply-id by} \text{metis} \)
  qed

lemma ptrm-alpha-equiv-reflexive:
  shows \( M \approx M \)
by \((\text{induction } M, \text{ auto simp add: ptrm-alpha-equiv.intros})\)

corollary ptrm-alpha-equiv-reflp:
  shows \( \text{reflp ptrm-alpha-equiv} \)
unfolding \( \text{reflp-def} \) using \( \text{ptrm-alpha-equiv-reflexive by auto} \)

lemma ptrm-alpha-equiv-symmetric:
  assumes \( X \approx Y \)
  shows \( Y \approx X \)
using \( \text{assms proof(induction rule: ptrm-alpha-equiv.induct, auto simp add: ptrm-alpha-eq} \text{ui-reflexive}) \)
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-swp-transfer ptrm-unit-commutes by metis

have \(b \notin \text{ptrm-fvs}\ A\) using \(\langle a \notin \text{ptrm-fvs}\ B \rangle\ \langle A \approx [a \leftrightarrow b] \cdot B \approx a \neq b \rangle\)

using ptrm-alpha-equiv-fvs-transfer by metis

show \(?\text{case}\ \langle b \neq a \rangle\ \langle B \approx [b \leftrightarrow a] \cdot A \rangle\ \langle b \notin \text{ptrm-fvs}\ A \rangle\)

using ptrm-alpha-equiv by metis

next

qed

corollary ptrm-alpha-equiv-symp:
shows \(\text{symp}\ \text{ptrm-alpha-equiv}\)

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 \(\text{assms}\) proof(induction size \(X\) arbitrary; \(X\ Y\ Z\) rule: less-induct)

fix \(X\ Y\ Z::\ 'a\ \text{ptrm}\)

assume IH: \(\forall K\ Y\ Z::\ 'a\ \text{ptrm}.\ \text{size}\ K < \text{size}\ X \implies K \approx Y \implies Y \approx Z \implies K \approx Z\)

assume \(X \approx Y\) and \(Y \approx Z\)

show \(X \approx Z\) proof(cases \(X\))

case \((\text{PUnit})\)

hence \(Y = \text{PUnit}\) using \(\langle X \approx Y \rangle\) \(\text{unitE}\) by metis

hence \(Z = \text{PUnit}\) using \(\langle Y \approx Z \rangle\) \(\text{unitE}\) by metis

thus \(?\text{thesis}\) using ptrm-alpha-equiv-/unit \(\langle X = \text{PUnit} \rangle\) by metis

next

case \((\text{PVar } x)\)

hence \(\text{PVar } x \approx Y\) using \(\langle X \approx Y \rangle\) by auto

hence \(Y = \text{PVar } x\) using \(\text{varE}\) by metis

hence \(\text{PVar } x \approx Z\) using \(\langle Y \approx Z \rangle\) by auto

hence \(Z = \text{PVar } x\) using \(\text{varE}\) by metis

thus \(?\text{thesis}\) using ptrm-alpha-equiv/var \(\langle X = \text{PVar } x \rangle\) by metis

next

case \((\text{PApp } A\ B)\)

obtain \(C\ D\) where \(Y = \text{PApp } C\ D\) and \(A \approx C\) and \(B \approx D\)

using \(\text{appE}\) \(\langle X = \text{PApp } A\ B \rangle\ \langle X \approx Y \rangle\) by metis

hence \(\text{PApp } C\ D \approx Z\) using \(\langle Y \approx Z \rangle\) by \(\text{simp}\)

from this obtain \(E\ F\) where \(Z = \text{PApp } E\ F\) and \(C \approx E\) and \(D \approx F\) using \(\text{appE}\) by metis

have \(\text{size } A < \text{size } X\) and \(\text{size } B < \text{size } X\) using \(\langle X = \text{PApp } A\ B \rangle\) by auto

hence \(A \approx E\) and \(B \approx F\) using \(\text{IH}\ \langle A \approx C \rangle\ \langle C \approx E \rangle\ \langle B \approx D \rangle\ \langle D \approx F \rangle\) by auto

thus \(?\text{thesis}\) using \(\langle X = \text{PApp } A\ B \rangle\ \langle Z = \text{PApp } E\ F \rangle\) ptrm-alpha-equiv/app
by metis

next

case (PFn x T A)
   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 ∧ A ≈ B) ∨ (x ≠ y ∧ A ≈ [x ↔ y] · B ∧ x ∉ ptrm-fvs B)

   using fnE ⟨X ≈ Y⟩ ⟨X = PFn x T A⟩ by metis
obtain z C where Z = PFn z T C
   and Z-cases: (y = z ∧ B ≈ C) ∨ (y ≠ z ∧ B ≈ [y ↔ z] · C ∧ y ∉ ptrm-fvs C)

   using fnE ⟨Y ≈ Z⟩ ⟨Y = PFn y T B⟩ by metis

consider
   x = y A ≈ B and y = z B ≈ C
   | x = y A ≈ B and y ≠ z B ≈ [y ↔ z] · C y ∉ ptrm-fvs C
   | x ≠ y A ≈ [x ↔ y] · B x ∉ ptrm-fvs B and y = z B ≈ C
   | x ≠ y A ≈ [x ↔ y] · B x ∉ ptrm-fvs B and y ≠ z B ≈ [y ↔ z] · C y ∉ ptrm-fvs C and x = z
   | x ≠ y A ≈ [x ↔ y] · B x ∉ ptrm-fvs B and y ≠ z B ≈ [y ↔ z] · C y ∉ ptrm-fvs C and x ≠ z
      using Y-cases Z-cases by auto

thus ?thesis proof(cases)
   case 1
      have A ≈ C using ⟨A ≈ B⟩ ⟨B ≈ C⟩ IH * by metis
      have x = z using ⟨x = y⟩ ⟨y = z⟩ by metis
      show ?thesis using ⟨A ≈ C⟩ ⟨x = z⟩ X Z
         using ptrm-alpha-equiv.fn1 by metis
   next
   case 2
      have x ≠ z using ⟨x = y⟩ ⟨y ≠ z⟩ by metis
      have A ≈ [x ↔ z] · C using ⟨A ≈ B⟩ ⟨B ≈ [y ↔ z] · C⟩ ⟨x = y⟩ IH *
         by metis
      have x ∉ ptrm-fvs C using ⟨x = y⟩ ⟨y ∉ ptrm-fvs C⟩ by metis
      thus ?thesis using ⟨x ≠ z⟩ ⟨A ≈ [x ↔ z] · C⟩ ⟨x ∉ ptrm-fvs C⟩ ⟨x ∈ X Z
         using ptrm-alpha-equiv.fn2 by metis
   next
   case 3
      have x ≠ z using ⟨x ≠ y⟩ ⟨y = z⟩ by metis
      have [x ↔ y] · B ≈ [x ↔ y] · C using ⟨B ≈ C⟩ ptrm-alpha-equiv-prm
         by metis
      have A ≈ [x ↔ z] · C
         using ⟨A ≈ [x ↔ y] · B⟩ ⟨[x ↔ y] · B ≈ [x ↔ y] · C⟩ ⟨y = z⟩ IH *
         by metis
      have x ∉ ptrm-fvs C using ⟨B ≈ C⟩ ⟨x ∉ ptrm-fvs B⟩ ptrm-alpha-equiv-fes
         by metis
thus \(?thesis\) using \((x \neq z) \cdot (A \approx [x \leftrightarrow z] \cdot C) \cdot x \notin \ptrm-fvs\ C\) \(X Z\)
using \(\ptrm-alpha-equiv.fn2\) by \(\text{metis}\)
next
case 4
have \([x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C\)
using \((B \approx [y \leftrightarrow z] \cdot C) \cdot \ptrm-alpha-equiv-prm\) by \(\text{metis}\)
hence \(A \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C\)
using \((A \approx [x \leftrightarrow y] \cdot B) \cdot \text{IH} * \) by \(\text{metis}\)
hence \(A \approx ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C\) using \(\ptrm-prm-apply-compose\) by \(\text{metis}\)
hence \(A \approx ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C\) using \(\ptrm-unit-involution\) by \(\text{metis}\)
hence \(A \approx C\) using \(\ptrm-prm-apply-id\) by \(\text{metis}\)
thus \(?thesis\) using \((x = z) \cdot (A \approx C) \cdot X Z\)
using \(\ptrm-alpha-equiv.fn1\) by \(\text{metis}\)
next
case 5
have \(x \notin \ptrm-fvs\ C\) proof
have \(\ptrm-fvs\ B = \ptrm-fvs\ ([y \leftrightarrow z] \cdot C)\)
using \(\ptrm-alpha-equiv-fvs\ \(B \approx [y \leftrightarrow z] \cdot C\)\) by \(\text{metis}\)
hence \(x \notin \ptrm-fvs\ ([y \leftrightarrow z] \cdot C)\) using \(\langle x \notin \ptrm-fvs\ B\rangle\) by \(\text{auto}\)
hence \(x \notin [y \leftrightarrow z] \cdot \{\$\}\) \(\ptrm-fvs\ C\) using \(\ptrm-prm-fvs\) by \(\text{metis}\)
consider \(z \in \ptrm-fvs\ C \setminus \{z\} \notin \ptrm-fvs\ C\) by \(\text{auto}\)
thus \(?thesis\) proof (cases)
case 1
hence \([y \leftrightarrow z] \cdot \{\$\}\) \(\ptrm-fvs\ C = \ptrm-fvs\ C \setminus \{z\} \cup \{y\}\)
using \(\ptrm-set-unit-action \ptrm-unit-commutes\)
and \((z \in \ptrm-fvs\ C \setminus \{y \notin \ptrm-fvs\ C\})\) by \(\text{metis}\)
hence \(x \notin \ptrm-fvs\ C \setminus \{z\} \cup \{y\}\) using \(\langle x \notin |y \leftrightarrow z| \cdot \{\$\}\ \ptrm-fvs\\) by \(\text{auto}\)
hence \(x \notin \ptrm-fvs\ C \setminus \{z\} \cdot \{\$\}\ \ptrm-fvs\ C\) by \(\text{auto}\)
thus \(?thesis\) using \((x \neq y)\) by \(\text{auto}\)
next
case 2
hence \([y \leftrightarrow z] \cdot \{\$\}\) \(\ptrm-fvs\ C = \ptrm-fvs\ C\) using \(\ptrm-set-unit-inaction\)
\(\langle y \notin \ptrm-fvs\ C\rangle\) by \(\text{metis}\)
thus \(?thesis\) using \((x \notin [y \leftrightarrow z] \cdot \{\$\}\ \ptrm-fvs\ C)\) by \(\text{auto}\)
next
qed

have \(A \approx [x \leftrightarrow z] \cdot C\) proof
have \(A \approx ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C\)
using \(\text{IH} * (A \approx [x \leftrightarrow y] \cdot B) \cdot (B \approx [y \leftrightarrow z] \cdot C)\)
and \(\ptrm-alpha-equiv-prm \ptrm-prm-apply-compose\) by \(\text{metis}\)
have \(([x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C \approx [x \leftrightarrow z] \cdot C\) proof

22
have $ds \ ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \ [x \leftrightarrow z] = \ {x, y}$
using $\langle x \neq y \rangle \ (y \neq z) \ (x \neq z)$ prm-disagreement-composition by metis

hence $\bigwedge a. \ a \in ds \ ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \ [x \leftrightarrow z] \implies a \notin \ ptrm-fus C$
using $\langle x \notin \ ptrm-fus C \rangle \ (y \notin \ ptrm-fus C)$,
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 $A \approx ([x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C \ (\ [x \leftrightarrow y] \circ [y \leftrightarrow z]) \cdot C \approx [x \leftrightarrow z] \cdot C$ by metis

qed

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

next

next

next

case $\ (PPair \ A \ B)$

obtain $C \ D$ where $Y = \ PPair \ C \ D$ and $A \approx C$ and $B \approx D$
using pairE $\langle X = \ PPair \ A \ B \rangle \ (X \approx Y)$ 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 $\langle X = PP\ A \ B \rangle$ by auto

hence $A \approx E$ and $B \approx F$ using $IH \ (A \approx C) \ (C \approx E) \ (B \approx D) \ (D \approx F)$ by auto

thus $?thesis$ using $\langle X = PPair \ A \ B \rangle \ (Z = \ PPair \ E \ F) \ ptrm-alpha-equiv.pair$ by metis

next

case $\ (PFst \ P)$

obtain $Q$ where $Y = \ PFst \ Q \ P \approx Q$ using fstE $\langle X = \ PFst \ P \rangle \ (X \approx Y)$ by metis

obtain $R$ where $Z = \ PFst \ R \ Q \approx R$ using $\langle Y = \ PFst \ Q \ (Y \approx Z) \rangle$ by metis

have $\ size \ P < \ size \ X$ using $\langle X = \ PFst \ P \rangle$ by auto

hence $P \approx R$ using $IH \ (P \approx Q) \ (Q \approx R)$ by metis

thus $?thesis$ using $\langle X = \ PFst \ P \rangle \ (Z = \ PFst \ R) \ ptrm-alpha-equiv.fst$ by metis

next

case $\ (PSnd \ P)$

obtain $Q$ where $Y = PSnd \ Q \ P \approx Q$ using sndE $\langle X = PSnd \ P \rangle \ (X \approx Y)$ by metis

obtain $R$ where $Z = PSnd \ R \ Q \approx R$ using $\langle Y = PSnd \ Q \ (Y \approx Z) \rangle$ by metis

23
have size P < size X using \( X = PSnd P \) by auto
hence P \( \approx \) R using IH \( Q \approx P \) \( \approx Q \approx R \) by metis
thus ?thesis using \( X = PSnd P \) \( Z = PSnd R \) ptrm-alpha-equiv.snd by metis
next
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 \( \rightarrow \) type

fun ptrm-infer-type :: 'a typing-ctx \( \Rightarrow \) 'a ptrm \( \Rightarrow \) type option where
\( \text{ptrm-infer-type} \) \( \Gamma \) PUnit = Some TUnit
\( \text{ptrm-infer-type} \) \( \Gamma \) (PVar x) = \( \Gamma \) x
\( \text{ptrm-infer-type} \) \( \Gamma \) (PApp A B) = (case (ptrm-infer-type \( \Gamma \) A, ptrm-infer-type \( \Gamma \) B) of
\( \text{Some} \) (TArr \( \tau \) \( \sigma \), \( \text{Some} \) \( \tau' \)) \Rightarrow (if \( \tau = \tau' \) then \( \text{Some} \) \( \sigma \) else None)
| \( \_ \Rightarrow \) None
)
\( \text{ptrm-infer-type} \) \( \Gamma \) (PFN x \( \tau \) A) = (case ptrm-infer-type (\( \Gamma \)(x \( \mapsto \) \( \tau \))) A of
\( \text{Some} \) \( \sigma \) \Rightarrow \text{Some} \ (TArr \( \tau \) \( \sigma \))
| None \Rightarrow None
)
\( \text{ptrm-infer-type} \) \( \Gamma \) (PPair A B) = (case (ptrm-infer-type \( \Gamma \) A, ptrm-infer-type \( \Gamma \) B) of
\( \text{Some} \) \( \tau \), \( \text{Some} \) \( \sigma \) \Rightarrow \text{Some} \ (TPair \( \tau \) \( \sigma \))
| \_ \Rightarrow \) None
)
\( \text{ptrm-infer-type} \) \( \Gamma \) (PFst P) = (case ptrm-infer-type \( \Gamma \) P of
\( \text{Some} \) (TPair \( \tau \) \( \sigma \)) \Rightarrow \text{Some} \( \tau \)
| \_ \Rightarrow \) None
)
\( \text{ptrm-infer-type} \) \( \Gamma \) (PSnd P) = (case ptrm-infer-type \( \Gamma \) P of
\( \text{Some} \) (TPair \( \tau \) \( \sigma \)) \Rightarrow \text{Some} \( \sigma \)
| \_ \Rightarrow \) None
)

lemma ptrm-infer-type-swp-types:
assumes a \( \neq \) b
shows ptrm-infer-type (\( \Gamma \)(a \( \mapsto \) T, b \( \mapsto \) S)) X = ptrm-infer-type (\( \Gamma \)(a \( \mapsto \) S, b \( \mapsto \) T)) ((a \( \leftrightarrow \) b) \cdot X)
using assms proof (induction X arbitrary: T S \( \Gamma \))
case \( \text{PUnit} \)
thus ?case by simp
next

24
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: prm-unit-action)
  next

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

  assume x ≠ a ∧ x ≠ b
  thus ?thesis by (simp add: prm-unit-inaction)
  next

  qed

next
case (PApp A B)
  thus ?case by simp
next
case (PFN x τ A)
  hence *:
    ptrm-infer-type (Γ(a ↦→ T, b ↦→ S)) A = ptrm-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
      ptrm-infer-type (Γ(a ↦→ S, b ↦→ T)) ([a ↔ b] · PFn x τ A)
    = ptrm-infer-type (Γ(a ↦→ S, b ↦→ T)) (PFN b τ ([a ↔ b] · A))
    using prm-unit-action ptrm-apply-prm.simps(4) by metis
    moreover have ... = (case ptrm-infer-type (Γ(a ↦→ S, b ↦→ τ)) ([a ↔ b] · A) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
      by simp
    moreover have ...
      = (case ptrm-infer-type (Γ(a ↦→ τ, b ↦→ S)) A of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
      using * by metis
    moreover have ...
      = (case ptrm-infer-type (Γ(b ↦→ S, a ↦→ τ)) A of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
      using (a ≠ b) fun-upd-twist fun-upd-upd by metis
    ultimately show ?thesis by metis
next
case 2
  hence
      ptrm-infer-type \((\Gamma(a \mapsto S, b \mapsto T))\) \([(a \leftrightarrow b) \cdot PFn x \tau A]\)
  ⇒ ptrm-infer-type \((\Gamma(a \mapsto S, b \mapsto T))\) \((PFn a \tau [(a \leftrightarrow b) \cdot A])\)
    using \(\text{prm-unit-action} \text{ prm-unit-commutes} \text{ ptrm-apply-prm.simps(4)}\) by \text{metis}
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto T \mapsto T)(a \mapsto \tau)) [(a \leftrightarrow b) \cdot A]\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    by simp
    moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto \tau, b \mapsto T)) [(a \leftrightarrow b) \cdot A]\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
      using \(\text{fun-upd-upd} \text{ fun-upd-twist} \(a \neq b\)\) by \text{metis}
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto \tau)) A\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    using \(\ast\) by \text{metis}
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto S)(b \mapsto \tau)) A\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    using \(\langle a \neq b \rangle\) \(\text{fun-upd-upd}\) by \text{metis}
  moreover have \(\ldots = (\text{ptrm-infer-type} \Gamma(b \mapsto S, a \mapsto T)) \((PFn x \tau A)\)\)
    using \(\langle x = b \rangle\) by simp
  moreover have \(\ldots = (\text{ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto S)) \((PFn x \tau A)\)\)
    using \(\langle a \neq b \rangle\) \(\text{fun-upd-twist}\) by \text{metis}
ultimately show \(?thesis\) by \text{metis}

next
case 3
  hence \(x \neq a\) \(x \neq b\) by \text{auto}
  hence
      \(\text{ptrm-infer-type} \Gamma(a \mapsto S, b \mapsto T)\) \([(a \leftrightarrow b) \cdot PFn x \tau A]\)
  ⇒ \(\text{ptrm-infer-type} \Gamma(a \mapsto S, b \mapsto T)\) \((PFn x \tau [(a \leftrightarrow b) \cdot A])\)
    by \(\text{simp add: ptrm-unit-inaction}\)
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto T)(x \mapsto \tau)) [(a \leftrightarrow b) \cdot A]\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    by simp
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(x \mapsto \tau, a \mapsto S, b \mapsto T)) [(a \leftrightarrow b) \cdot A]\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    using \(\langle x \neq a \rangle\) \(\langle x \neq b \rangle\) \(\text{fun-upd-twist}\) by \text{metis}
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(x \mapsto \tau, a \mapsto T, b \mapsto S)) A\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    using \(\ast\) by \text{metis}
  moreover have \(\ldots = (\text{case} \text{ ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto S, x \mapsto \tau)) A\) of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
    using \(\langle x \neq a \rangle\) \(\langle x \neq b \rangle\) \(\text{fun-upd-twist}\) by \text{metis}
  moreover have \(\ldots = (\text{ptrm-infer-type} \Gamma(a \mapsto T, b \mapsto S)) \((PFn x \tau A)\)\)
    using \(\text{simp}\)
ultimately show \(?thesis\) by \text{metis}
next
qed
next

26
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 ≠ b b /∈ ptrm-fvs X
  shows ptrm-infer-type (Γ(a ↦→ τ)) X = ptrm-infer-type (Γ(b ↦→ τ)) ([a ↔ b] · X)
  using assms proof (induction X arbitrary: τ Γ)
  case (PUnit)
  thus ?case by simp
next
case (PVar x)
  hence x ≠ b by simp
  consider x = a | x ≠ a by auto
  thus ?case proof (cases)
    case 1
    hence [a ↔ b] · (PVar x) = PVar b
    and ptrm-infer-type (Γ(a ↦→ τ)) (PVar x) = Some τ using prm-unit-action
    by auto
    thus ?thesis by auto
  next

case 2
  hence *: [a ↔ b] · PVar x = PVar x using (x ≠ b) prm-unit-inaction by simp
  consider ∃σ. Γ x = Some σ | Γ x = None by auto
  thus ?thesis proof (cases)
    assume ∃σ. Γ x = Some σ
    from this obtain σ where Γ x = Some σ by auto
    thus ?thesis using * (x ≠ a) (x ≠ b) by auto
  next
  assume Γ x = None
  thus ?thesis using * (x ≠ a) (x ≠ b) by auto
qed
next

next
case (PApp A B)
  have b /∈ ptrm-fvs A and b /∈ ptrm-fvs B using PApp.prems by auto
  hence ptrm-infer-type (Γ(a ↦→ τ)) A = ptrm-infer-type (Γ(b ↦→ τ)) ([a ↔ b] ·
A) 
and $\text{ptrm-infer-type}(\Gamma(a \mapsto \tau)) \cdot B = \text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot ([a \leftrightarrow b] \cdot B)$

using $\text{PApp.IH \ assms\ by\ metis}$

thus $\text{?case\ by\ (metis\ \text{ptrm-apply-prm.simps}(3)\ \text{ptrm-infer-type.simps}(3))}$

next case (PFN $x \mapsto \sigma A$

consider $b \neq x \land b \not\in \text{ptrm-fvs\ } A \mid b = x$ using $\text{PFN.prems\ by\ auto}$

thus $\text{?case\ proof(cases)}$

case 1

hence $b \neq x \land b \not\in \text{ptrm-fvs\ } A$ by auto

hence $\vdash \forall \tau. \text{ptrm-infer-type}(\Gamma(a \mapsto \tau)) \cdot A = \text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot ([a \leftrightarrow b] \cdot A)$

using $\text{PFN.IH\ assms\ by\ metis}$

consider $a = x \mid a \neq x$ by auto

thus $\text{?thesis\ proof(cases)}$

case 1

hence $\text{ptrm-infer-type}(\Gamma(a \mapsto \tau)) \cdot (PFN \ a \mapsto \sigma A) = \text{ptrm-infer-type}(\Gamma(a \mapsto \tau)) \cdot (PFN \ a \mapsto \sigma A)$

and

$\text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot ([a \leftrightarrow b] \cdot PFN \ x \mapsto \sigma A) = 
\text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot (PFN \ b \mapsto \sigma ([a \leftrightarrow b] \cdot A))$

by (auto simp add: \text{prm-unit-action})

thus $\text{?thesis\ using\ *\ \text{ptrm-infer-type.simps}(4)\ \text{fun-upd-upd\ by\ metis}$

next

case 2

hence

$\text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot ([a \leftrightarrow b] \cdot PFN \ x \mapsto \sigma A) = 
\text{ptrm-infer-type}(\Gamma(b \mapsto \tau)) \cdot (PFN \ b \mapsto \sigma ([a \leftrightarrow b] \cdot A))$

by (simp add: \text{prm-unit-inaction})

moreover have $\ldots = (\text{case\ ptrm-infer-type}(\Gamma(b \mapsto \tau, x \mapsto \sigma)) \cdot ([a \leftrightarrow b] \cdot A) \cdot A$ of None $\Rightarrow$ None $\mid$ Some $\sigma' \Rightarrow$ Some ($\text{TArr\ } \sigma\ \sigma'\ )$

by simp

moreover have $\ldots = (\text{case\ ptrm-infer-type}(\Gamma(x \mapsto \sigma, b \mapsto \tau)) \cdot ([a \leftrightarrow b] \cdot A) \cdot A$ of None $\Rightarrow$ None $\mid$ Some $\sigma' \Rightarrow$ Some ($\text{TArr\ } \sigma\ \sigma'\ )$

using ($b \neq x$) \text{fun-upd-twist\ by\ metis}

moreover have $\ldots = (\text{case\ ptrm-infer-type}(\Gamma(x \mapsto \sigma, a \mapsto \tau)) \cdot A$ of None $\Rightarrow$ None $\mid$ Some $\sigma' \Rightarrow$ Some ($\text{TArr\ } \sigma\ \sigma'\ )$

using * by metis

moreover have $\ldots = (\text{case\ ptrm-infer-type}(\Gamma(a \mapsto \tau, x \mapsto \sigma)) \cdot A$ of None $\Rightarrow$ None $\mid$ Some $\sigma' \Rightarrow$ Some ($\text{TArr\ } \sigma\ \sigma'\ )$

using ($a \neq x$) \text{fun-upd-twist\ by\ metis}

moreover have $\ldots = \text{ptrm-infer-type}(\Gamma(a \mapsto \tau)) \cdot (PFN \ x \mapsto \sigma A)$

by simp

ultimately show $\text{?thesis\ by\ metis}$

next

qed
next

case 2

hence \( a \neq x \) using assms by auto

have

\[
\text{ ptrm-infer-type } \left( \Gamma(a \mapsto \tau)(b \mapsto \sigma) \right) A = \\
\text{ ptrm-infer-type } \left( \Gamma(b \mapsto \tau)(a \mapsto \sigma) \right) ([a \leftrightarrow b] \cdot A)
\]

using ptrm-infer-type-swp-types using \( a \neq b \) fun-upd-twist by metis

thus ?thesis

using \( b = x \) prm-unit-action prm-unit-commutes

using ptrm-infer-type.simps(4) ptrm-apply-prm.simps(4) by metis

next

qed

next
case \((\text{PPair } A B)\)

thus ?case by simp

next
case \((\text{PFst } P)\)

thus ?case by simp

next
case \((\text{PSnd } P)\)

thus ?case by simp

next

qed


lemma ptrm-infer-type-alpha-equiv:

assumes \( X \approx Y \)

shows \( \text{ ptrm-infer-type } \Gamma X = \text{ ptrm-infer-type } \Gamma Y \)

using assms proof(induction arbitrary: \( \Gamma \))

case \((\text{fn2 } a \ b \ A \ B \ T \ \Gamma)\)

hence \( \text{ ptrm-infer-type } \left( \Gamma(a \mapsto T) \right) A = \text{ ptrm-infer-type } \left( \Gamma(b \mapsto T) \right) B \)

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 \('a \ trm = 'a \ ptrm \ / \ ptrm-alpha-equiv\)

proof(rule equivpI)

  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 :: \('a \ trm \ is \ PUnit\).
lift-definition \textit{Var} :: \( 	exttt{'}a \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PVar} \).

lift-definition \textit{App} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PApp} \) using \( \texttt{ptrm-alpha-equiv} \) app.

lift-definition \textit{Fn} :: \( 	exttt{'}a \Rightarrow \text{ type} \Rightarrow \texttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PFN} \) using \( \texttt{ptrm-alpha-equiv} \) fn1.

lift-definition \textit{Pair} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PPair} \) using \( \texttt{ptrm-alpha-equiv} \).

lift-definition \textit{Fst} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PFst} \) using \( \texttt{ptrm-alpha-equiv} \).

lift-definition \textit{Snd} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{PSnd} \) using \( \texttt{ptrm-alpha-equiv} \).

lift-definition \textit{fvs} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ set} \) is \( \texttt{ptrm-fvs} \) using \( \texttt{ptrm-alpha-equiv-fvs} \).

lift-definition \textit{prm} :: \( 	exttt{'}a \text{ prm} \Rightarrow \texttt{'}a \text{ trm} \Rightarrow \texttt{'}a \text{ trm} \) is \( \texttt{ptrm-apply-prm} \) using \( \texttt{ptrm-alpha-equiv-prm} \).

lift-definition \textit{depth} :: \( 	exttt{'}a \text{ trm} \Rightarrow \texttt{nat} \) is \( \texttt{size} \) using \( \texttt{ptrm-size-alpha-equiv} \).

lemma \textit{depth-prm}:
  shows \( \text{depth} \ (\pi \cdot A) = \text{depth} A \)
by (transfer, \texttt{metis \_ptrm-size-prm})

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

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

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

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

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

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

lemma \textit{unit-not-app}:
  shows \( \text{Unit} \neq \text{App} A B \)

\section*{30}
proof (transfer)
  fix A B :: 'a ptrm
  show \neg P\text{Unit} \approx P\text{App} A B
proof (rule classical)
  assume \neg \neg P\text{Unit} \approx P\text{App} A B
  hence False using unitE by fastforce
  thus ?thesis by blast
qed

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

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

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

lemma unit-not-snd:
  shows Unit \neq P\text{Snd} P
proof (transfer)
fix \( P :: 'a \text{ ptrm} \)
show \( \neg \text{PUnit} \approx \text{PSnd} \ P \)
proof (rule classical)
  assume \( \neg \neg \text{PUnit} \approx \text{PSnd} \ P \)
  hence False using unitE by fastforce
  thus \( \neg \thesis \) by blast
qed

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

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

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

lemma var-not-fst:
  shows \( \text{Var} \ x \neq \text{Fst} \ P \)
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

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

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

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

lemma app-not-fst:

shows \( App \ A \ B \neq Fst \ P \)

proof (transfer)

fix \( A \ B \ P :: 'a \) ptrm

show \( \neg PApp \ A \ B \approx PFst \ P \)

qed
proof (rule classical)
  assume \( \neg\neg P\text{App} A B \approx PF\text{st} P \)
  hence False using appE by auto
  thus \( ?\text{thesis} \) by blast
qed

lemma app-not-snd:
shows App A B \( \neq \) Snd P
proof (transfer)
  fix A B P :: 'a pterm
  show \( \neg P\text{App} A B \approx P\text{Snd} P \)
  proof (rule classical)
    assume \( \neg\neg P\text{App} A B \approx P\text{Snd} P \)
    hence False using appE by auto
    thus \( ?\text{thesis} \) 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 \( \neg P\text{Fn} x T A \approx P\text{Pair} C D \)
  proof (rule classical)
    assume \( \neg\neg P\text{Fn} x T A \approx P\text{Pair} C D \)
    hence False using fnE by fastforce
    thus \( ?\text{thesis} \) 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 \( \neg P\text{Fn} x T A \approx PF\text{st} P \)
  proof (rule classical)
    assume \( \neg\neg P\text{Fn} x T A \approx PF\text{st} P \)
    hence False using fnE by fastforce
    thus \( ?\text{thesis} \) 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 \( \neg P\text{Fn} x T A \approx P\text{Snd} P \)
  proof (rule classical)
assume \( \neg\text{PFN } x \ T A \approx \text{PSnd } P \)
hence False using \( \text{fnE} \) by \( \text{fastforce} \)
thus \(?\text{thesis} \) by \( \text{blast} \)
qed

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

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

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

lemma trm-simp:
shows
\( \text{Var } x = \text{Var } y \implies x = y \)
\( \text{App } A \ B = \text{App } C \ D \implies A = C \)
\( \text{App } A \ B = \text{App } C \ D \implies B = D \)
\( \text{Fn } x \ T \ A = \text{Fn } y \ S \ B \implies \)
\( (x = y \land T = S \land A = B) \lor (x \neq y \land T = S \land x \notin \text{fvs } B \land A = [x \leftrightarrow y]) \).

35
\( \text{Pair } A \; B = \text{Pair } C \; D \implies A = C \)  
\( \text{Pair } A \; B = \text{Pair } C \; D \implies B = D \)  
\( \text{Fst } P = \text{Fst } Q \implies P = Q \)  
\( \text{Snd } P = \text{Snd } Q \implies P = Q \)

**proof**

- show \( \text{Var } x = \text{Var } y \implies x = y \) by (transfer, insert ptrm.inject varE, fastforce)
- show \( \text{App } A \; B = \text{App } C \; D \implies A = C \) by (transfer, insert ptrm.inject appE, auto)
- show \( \text{App } A \; B = \text{App } C \; D \implies B = D \) by (transfer, insert ptrm.inject appE, auto)
- show \( \text{Pair } A \; B = \text{Pair } C \; D \implies A = C \) by (transfer, insert ptrm.inject pairE, auto)
- show \( \text{Pair } A \; B = \text{Pair } C \; D \implies B = D \) by (transfer, insert ptrm.inject pairE, auto)
- show \( \text{Fst } P = \text{Fst } Q \implies P = Q \) by (transfer, insert ptrm.inject fstE, auto)
- show \( \text{Snd } P = \text{Snd } Q \implies P = Q \) by (transfer, insert ptrm.inject sndE, auto)

**proof**

\( (x = y \land T = S \land A = B) \lor (x \neq y \land T = S \land x \notin \text{ptrm-fvs } B \land A = [x \leftrightarrow y] \cdot B) \)

**proof**

\( \text{fix } x \; y :: \text{'a and } T \; S :: \text{type and } A \; B :: \text{'a ptrm} \)  
\( \text{assume } \ast : \text{PFN } x \; T \; A \approx \text{PFN } y \; S \; B \)  
\( \text{thus } x = y \land T = S \land A = B \lor x \neq y \land T = S \land x \notin \text{ptrm-fus } B \land A \approx [x \leftrightarrow y] \cdot B \)  

- case (\( 2 \; C \))
  - thus \( \ast \text{case by simp} \)
- next
- case (\( 3 \; z \; C \))
  - thus \( \ast \text{case by simp} \)
- next
- qed
- qed

**lemma fn-eq:**

- assumes \( x \neq y \land x \notin \text{ptrm-fus } B \land A = [x \leftrightarrow y] \cdot B \)
- shows \( \text{Fnt } x \; T \; A = \text{Fnt } y \; T \; B \)  
- using assms by (transfer', metis ptrm-alpha-equiv.fn2)

**lemma trm-prm-simp:**

- shows  
  \( \pi \cdot \text{Unit} = \text{Unit} \)  
  \( \pi \cdot \text{Var } x = \text{Var } (\pi \cdot x) \)  
  \( \pi \cdot \text{App } A \; B = \text{App } (\pi \cdot A) \; (\pi \cdot B) \)  
  \( \pi \cdot \text{Fnt } x \; T \; A = \text{Fnt } (\pi \cdot x) \; T \; (\pi \cdot A) \)  
  \( \pi \cdot \text{Pair } A \; B = \text{Pair } (\pi \cdot A) \; (\pi \cdot B) \)
\[ \pi \cdot \text{Fst} \ P = \text{Fst} \ (\pi \cdot P) \]
\[ \pi \cdot \text{Snd} \ P = \text{Snd} \ (\pi \cdot P) \]

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 \[ \pi \cdot \sigma \cdot A = (\pi \circ \sigma) \cdot A \]
by (transfer', metis ptrm-prm-apply-compose ptrm-alpha-equiv-reflexive)

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

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

lemma var-prm-action:
shows \[ [a \leftrightarrow b] \cdot \text{Var} \ a = \text{Var} \ b \]
by (transfer', simp add: ptrm-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: ptrm-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 \[ a \notin \text{fes} \ X \ b \notin \text{fes} \ X \]
shows \[ [a \leftrightarrow b] \cdot X = X \]
using assms by (transfer', metis ptrm-prm-unit-inaction)

lemma trm-prm-agreement-equiv:
assumes \[ \forall a. \ a \in \text{ds} \ \pi \ \sigma \implies a \notin \text{fes} \ M \]
shows \[ \pi \cdot M = \sigma \cdot M \]
using assms by (transfer, metis ptrm-prm-agreement-equiv)
**lemma** trm-induct:

**fixes** \( P :: 'a \text{ trm} \Rightarrow \text{bool} \)

**assumes**
- \( P \ Unit \)
- \( \forall x. \ P (\Var x) \)
- \( \forall A. B. \ [P A; P B] \Rightarrow P (\App A B) \)
- \( \forall x. T. A. P A \Rightarrow P (\Fst x A) \)
- \( \forall A. B. [P A; P B] \Rightarrow P (\Pair A B) \)
- \( \forall A. P A \Rightarrow P (\Fn x T A) \)
- \( \forall A. P A \Rightarrow P (\Snd A) \)

**shows** \( P M \)

**proof**

- **have** \( \forall X. P (\abs-trm X) \)

**proof** (rule ptrm.induct)

- **show** \( P (\abs-trm \\text{PU}nit) \)
  - **using** \( \text{assms}(1) \ Unit.\abs-eq \) by metis

- **show** \( P (\abs-trm (\PVar x)) \) for \( x \)
  - **using** \( \text{assms}(2) \ Var.\abs-eq \) by metis

- **show** \( [P (\abs-trm \text{A}); P (\abs-trm \text{B})] \Rightarrow P (\abs-trm (\PApp A B)) \) for \( A B \)
  - **using** \( \text{assms}(3) \ App.\abs-eq \) by metis

- **show** \( P (\abs-trm \text{A}) \Rightarrow P (\abs-trm (\PFn x T A)) \) for \( x T A \)
  - **using** \( \text{assms}(4) \ Fn.\abs-eq \) by metis

- **show** \( [P (\abs-trm \text{A}); P (\abs-trm \text{B})] \Rightarrow P (\abs-trm (\PPair A B)) \) for \( A B \)
  - **using** \( \text{assms}(5) \ \Pair.\abs-eq \) by metis

- **show** \( P (\abs-trm \text{A}) \Rightarrow P (\abs-trm (\PFst A)) \) for \( A \)
  - **using** \( \text{assms}(6) \ \Fst.\abs-eq \) by metis

- **show** \( P (\abs-trm \text{A}) \Rightarrow P (\abs-trm (\PSnd A)) \) for \( A \)
  - **using** \( \text{assms}(7) \ \Snd.\abs-eq \) by metis

**qed**

- **thus** \( \?\text{thesis} \) using trm.abs-induct by auto

**qed**

**lemma** trm-cases:

**assumes**
- \( M = \text{Unit} \Rightarrow P M \)
- \( \forall x. M = \Var x \Rightarrow P M \)
- \( \forall A. B. M = \App A B \Rightarrow P M \)
- \( \forall T A. M = \Fn x T A \Rightarrow P M \)
- \( \forall A. B. M = \Pair A B \Rightarrow P M \)
- \( \forall A. M = \Fst A \Rightarrow P M \)
- \( \forall A. M = \Snd A \Rightarrow P M \)

**shows** \( P M \)

**using** \( \text{assms} \) by (induction rule: trm-induct, auto)

**lemma** trm-depth-induct:

**assumes**
- \( P \ Unit \)
- \( \forall x. P (\Var x) \)
- \( \forall A. B. [\forall K. \text{depth} K < \text{depth} (\App A B) \Rightarrow P K] \Rightarrow P (\App A B) \)
\( \forall M \cdot (\forall K. \text{depth} K < \text{depth} (\text{Fn} x T A) \Rightarrow P K) \Rightarrow P (\text{Fn} x T A) \)
\( \forall A B. (\forall K. \text{depth} K < \text{depth} (\text{Pair} A B) \Rightarrow P K) \Rightarrow P (\text{Pair} A B) \)
\( \forall A. (\forall K. \text{depth} K < \text{depth} (\text{Fst} A) \Rightarrow P K) \Rightarrow P (\text{Fst} A) \)
\( \forall A. (\forall K. \text{depth} K < \text{depth} (\text{Snd} A) \Rightarrow P K) \Rightarrow P (\text{Snd} A) \)

shows \( P M \)

**proof** (induction depth \( M \) arbitrary; \( M \) rule: less-induct)

fix \( M :: \text{'a trm} \)
assume \( \text{IH: depth} K < \text{depth} M \Rightarrow P K \) for \( K \)

hence
\( M = \text{Unit} \Rightarrow P M \)
\( \forall x. \quad M = \text{Var} x \Rightarrow P M \)
\( \forall x T A. \quad M = \text{Fn} x T A \Rightarrow P M \)
\( \forall A B. \quad M = \text{Pair} A B \Rightarrow P M \)
\( \forall A. \quad M = \text{Fst} A \Rightarrow P M \)
\( \forall A. \quad M = \text{Snd} A \Rightarrow P M \)

using assms by blast

thus \( P M \) using \( \text{trm-cases} \) [where \( M = M \)] by blast

qed

context fresh begin

**lemma** fresh-fn:

fixes \( x :: \text{'a} \) and \( S :: \text{'a set} \)
assumes finite \( S \)
shows \( \exists y B. \; y \not\in S \land B = [y \leftrightarrow x] \cdot A \land (\text{Fn} x T A = \text{Fn} y T B) \)

**proof** –

have \(*: finite (\{x\} \cup \text{fvs} A \cup S)\) using fvs-finite assms by auto

obtain \( y \) where \( y = \text{fresh-in} (\{x\} \cup \text{fvs} A \cup S)\) by auto

hence \( y \not\in (\{x\} \cup \text{fvs} A \cup S)\) using fresh-axioms * unfolding class.fresh-def by metis

hence \( y \neq x \) \( y \not\in \text{fvs} A \) \( y \not\in S \) by auto

obtain \( B \) where \( B = [y \leftrightarrow x] \cdot A\) by auto

hence \( \text{Fn} x T A = \text{Fn} y T B \) using fn-eq \( \langle y \neq x \rangle \) \( \langle y \not\in \text{fvs} A \rangle\) by metis

thus \( \text{thesis} \) using \( \langle y \neq x \rangle \) \( \langle y \not\in S \rangle \) \( B\) by metis

qed

**lemma** trm-strong-induct:

fixes \( P :: \text{'a set} \Rightarrow \text{'a trm} \Rightarrow \text{bool} \)
assumes
\( \forall S \text{Unit} \)
\( \forall x. \quad P S (\text{Var} x) \)
\( \forall A B. \quad [P S A; P S B] \Rightarrow P S (\text{App} A B) \)
\( \forall x T. \quad x \not\in S \Rightarrow (\forall A. \quad P S A \Rightarrow P S (\text{Fn} x T A)) \)
\( \forall A B. \quad [P S A; P S B] \Rightarrow P S (\text{Pair} A B) \)
\( \forall A. \quad P S A \Rightarrow P S (\text{Fst} A) \)
\( \forall A. \quad P S A \Rightarrow P S (\text{Snd} A) \)

finite \( S \)

39
shows $PSM$

proof –

have $\forall \pi. \; PS(\pi \cdot M)$

proof (induction $M$ rule: $\text{trm-induct}$)

  case 1
  
  thus $?case \; \text{using} \; \text{assms}(1) \; \text{trm-prm-simp}(1)$ by metis

  next

  case (2 $x$)
  
  thus $?case \; \text{using} \; \text{assms}(2) \; \text{trm-prm-simp}(2)$ by metis

  next

  case (3 $A \; B$)
  
  thus $?case \; \text{using} \; \text{assms}(3) \; \text{trm-prm-simp}(3)$ by metis

  next

  case (4 $x \; T \; A$)
  
  have $\text{finite} \; S \; \text{finite} \; (\text{fvs}(\pi \cdot A)) \; \text{finite} \; \{\pi \; \$ \; x\}$
  
  using $\text{finite} \; S \; \text{fus-finite}$ by auto
  
  hence $\text{finite} \; (S \; \cup \; \text{fvs}(\pi \cdot A) \; \cup \; \{\pi \; \$ \; x\})$ by auto

  obtain $y$ where $y = \text{fresh-in} \; (S \; \cup \; \text{fus}(\pi \cdot A) \; \cup \; \{\pi \; \$ \; x\})$ by auto

  hence $y \notin S \; \cup \; \text{fus}(\pi \cdot A) \; \cup \; \{\pi \; \$ \; x\}$ using $\text{fresh-axioms}$ unfolding $\text{class.fresh-def}$

  using $\text{finite} \; (S \; \cup \; \text{fus}(\pi \cdot A) \; \cup \; \{\pi \; \$ \; x\})$ by metis

  hence $y \neq \pi \; \$ \; x \; y \notin \text{fus}(\pi \cdot A) \; y \notin S$ by auto

  hence $\ast \; \forall A. \; PS \; A \implies PS (\text{Fn} \; y \; T \; A)$ using $\text{assms}(4)$ by metis

  have $PS \; (\{y \leftrightarrow \pi \; \$ \; x\} \odot (\pi \cdot A))$ using $\textbf{4}$ by metis

  hence $PS \; (\text{Fn} \; y \; T \; (((y \leftrightarrow \pi \; \$ \; x) \odot (\pi \cdot A)))$ using $\ast$ by metis

  moreover have $(\text{Fn} \; y \; T \; (((y \leftrightarrow \pi \; \$ \; x) \odot (\pi \cdot A))) = \text{Fn} \; (\pi \; \$ \; x) \; T \; (\pi \cdot A)$

  using $\text{trm-prm-apply-compose \; fn-eq \; y \neq \pi \; \$ \; x \; y \notin \text{fus}(\pi \cdot A)$ by metis

  ultimately show $?case \; \text{using} \; \text{trm-prm-simp}(4)$ by metis

  next

  case (5 $A \; B$)
  
  thus $?case \; \text{using} \; \text{assms}(5) \; \text{trm-prm-simp}(5)$ by metis

  next

  case (6 $A$)
  
  thus $?case \; \text{using} \; \text{assms}(6) \; \text{trm-prm-simp}(6)$ by metis

  next

  case (7 $A$)
  
  thus $?case \; \text{using} \; \text{assms}(7) \; \text{trm-prm-simp}(7)$ by metis

  next

  qed

  hence $PS (\varepsilon \cdot M)$ by metis

  thus $PS \; M$ using $\text{trm-prm-apply-id}$ by metis

  qed

lemma $\text{trm-strong-cases}$:

  fixes $P :: 'a \; \text{set} \Rightarrow 'a \; \text{trm} \Rightarrow \text{bool}$

  assumes $M = \text{Unit} \implies PS \; M$
\( \forall x.\ M = \text{Var } x \implies P S M \)
\( \forall A \ B.\ M = \text{App } A \ B \implies P S M \)
\( \forall x \ T A.\ M = \text{Fn } x \ T A \implies x \notin S \implies P S M \)
\( \forall A \ B.\ M = \text{Pair } A \ B \implies P S M \)
\( \forall A.\ M = \text{Fst } A \implies P S M \)
\( \forall A.\ M = \text{Snd } A \implies P S M \)

\[ \text{finite } S \]

shows \( P S M \)
using \text{assms by (induction } S M \text{ rule: trm-strong-induct,metis+)}

**Lemma** \text{trm-strong-depth-induct}:

fixes \( P :: 'a \text{ set } \Rightarrow 'a \text{ trm } \Rightarrow \text{bool} \)

assumes
\[ P S \text{ Unit} \]
\[ \forall x.\ P S (\text{Var } x) \]
\[ \forall A \ B.\ \left[ \left[ \forall K.\ \text{depth } K < \text{depth } (\text{App } A \ B) \implies P S K \right] \implies P S (\text{App } A \ B) \right) \]
\[ \forall x \ T A.\ M = \text{Fn } x \ T A \implies x \notin S \implies P S M \]
\[ \forall A \ B.\ M = \text{Pair } A \ B \implies P S M \]
\[ \forall A.\ M = \text{Fst } A \implies P S M \]
\[ \forall A.\ M = \text{Snd } A \implies P S M \]

\[ \text{finite } S \]

shows \( P S M \)
proof (induction \( \text{depth } M \) arbitrary: \( M \) rule: less-induct)

fix \( M :: 'a \text{ trm} \)

assume \( \text{IH}: \text{depth } K < \text{depth } M \implies P S K \) for \( K \)

hence
\[ M = \text{Unit } \implies P S M \]
\[ \forall x.\ M = \text{Var } x \implies P S M \]
\[ \forall A \ B.\ M = \text{App } A \ B \implies P S M \]
\[ \forall x \ T A.\ M = \text{Fn } x \ T A \implies x \notin S \implies P S M \]
\[ \forall A \ B.\ M = \text{Pair } A \ B \implies P S M \]
\[ \forall A.\ M = \text{Fst } A \implies P S M \]
\[ \forall A.\ M = \text{Snd } A \implies P S M \]

\[ \text{finite } S \]

using assms \( \text{IH} \) by metis+

thus \( P S M \) using \text{trm-strong-cases}[\text{where } M=M] \ by blast

qed

**Lemma** \text{trm-prm-fvs}:

shows \( \text{fvs } (\pi \cdot M) = \pi \{S\} \text{ fvs } M \)

by (transfer, metis \text{ptrm-prm-fvs})

**Inductive** \text{typing} :: \( 'a \text{ typing-ctx } \Rightarrow 'a \text{ trm } \Rightarrow \text{type } \Rightarrow \text{bool } (-\vdash - : -) \) where
\( \text{tunit}: \Gamma \vdash \text{Unit } : T\text{Unit} \)
\( \text{tvar}: \Gamma \vdash \text{x = Some } \tau \implies \Gamma \vdash \text{Var } x : \tau \)
\( \text{tapp}: \Gamma \vdash f : (T\text{Arr } \tau \sigma); \Gamma \vdash x : \tau \implies \Gamma \vdash \text{App } f x : \sigma \)
\( \text{tfn}: \Gamma (x \mapsto \tau) \vdash A : \sigma \implies \Gamma \vdash \text{Fn } x \tau A : (T\text{Arr } \tau \sigma) \)
\( \text{tpair}: \Gamma \vdash A : \tau; \Gamma \vdash B : \sigma \implies \Gamma \vdash \text{Pair } A \ B : (T\text{Pair } \tau \sigma) \)
lemma typing-prm:
assumes \( \Gamma \vdash M : \tau \) \( \land \forall y. \ y \in \text{fvs} \ M \implies \Gamma \ y = \Delta \ (\pi \ y) \)
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 \( y \in \text{fvs} \ A \implies (\Gamma(x \mapsto \tau)) \ y = (\Delta(\pi \ x \mapsto \tau)) \ (\pi \ y) \) for \( y \)
proof (cases \( y = x \))
  case True
  thus ?thesis using fun-upd-apply by simp
next
case False
  assume \( y \in \text{fvs} \ A \)
  hence \( y \in \text{fvs} \ (\text{Fn} \ x \ \tau \ A) \) using fvs-simp (4) \( \langle y \neq x \rangle \) DiffI singletonD by fastforce
  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 \( \Gamma(a \mapsto \sigma) \vdash M : \tau \) \( b \notin \text{fvs} \ M \)
shows \( \Gamma(b \mapsto \sigma) \vdash [a \leftrightarrow b] \cdot M : \tau \)
proof -
  have \( y \in \text{fvs} \ M \implies (\Gamma(a \mapsto \sigma)) \ y = (\Gamma(b \mapsto \sigma)) \ ([a \leftrightarrow b] \ y) \) for \( y \)
proof
  assume \( y \in \text{fvs } M \)
  hence \( y \neq b \) using \( \text{assms(2)} \) 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 \( \text{cases, simp add: prm-unit-action, simp add: prm-unit-inaction} \ \langle y \neq b \rangle \)
  qed
  thus \(?thesis \) using \( \text{typing-prm} \ \text{assms(1)} \) by metis
  qed

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

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

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

lemma \( \text{typing-fnE} \):
  assumes \( \Gamma \vdash \text{Fn } x \ T \ A : \vartheta \)
  shows \( \exists \tau. \ \vartheta = (T\text{Arr } T \sigma) \land (\Gamma(x \mapsto T) \vdash A : \sigma) \)
using \( \text{assms} \) \( \text{proof(cases)} \)
  case \((\text{fn } y \ S \ B \ \sigma) \)
    from \( \text{this} \) consider
\( x = y \land T = S \land A = B \mid x \neq y \land T = S \land x \notin fvs B \land A = [x \leftrightarrow y] \cdot B \)
using \( \text{trm-simp}(4) \) by \( \text{metis} \)
thus \( \text{thesis} \)
\begin{proof}(cases)\\
\text{case 1}\\
\text{thus} \ ?\text{thesis} \text{ using } \text{tfn} \text{ by } \text{metis}\\
\text{next}\\
\text{case 2}\\
\text{thus} \ ?\text{thesis} \text{ using } \text{tfn} \text{ typing-swpr } \text{prm-unit-commutes} \text{ by } \text{metis}\\
\text{next}\\
\text{qed}\\
\text{next}\\
\text{qed (}
\text{metis} \text{ unit-not-fn},\\
\text{metis} \text{ var-not-fn},\\
\text{metis} \text{ app-not-fn},\\
\text{metis} \text{ fn-not-pair},\\
\text{metis} \text{ fn-not-fst},\\
\text{metis} \text{ fn-not-snd}\\
\text{)}
\end{proof}

\textbf{lemma} \( \text{typing-pairE} \):
\begin{proof}(cases)\\
\text{case } (\text{tpair } A \tau B \sigma)\\
\text{thus} \ ?\text{thesis} \text{ using } \text{trm-simp}(5) \text{ trm-simp}(6) \text{ by } \text{blast}\\
\text{next}\\
\text{qed (}
\text{metis} \text{ unit-not-pair},\\
\text{metis} \text{ var-not-pair},\\
\text{metis} \text{ app-not-pair},\\
\text{metis} \text{ fn-not-pair},\\
\text{metis} \text{ fn-not-fst},\\
\text{metis} \text{ fn-not-snd}\\
\text{)}
\end{proof}

\textbf{lemma} \( \text{typing-fstE} \):
\begin{proof}(cases)\\
\text{case } (\text{tfst } P \sigma)\\
\text{thus} \ ?\text{thesis} \text{ using } \text{trm-simp}(7) \text{ by } \text{blast}\\
\text{next}\\
\text{qed (}
\text{metis} \text{ unit-not-fst},\\
\text{metis} \text{ var-not-fst},\\
\text{metis} \text{ app-not-fst},\\
\text{metis} \text{ fn-not-fst},\\
\text{metis} \text{ pair-not-fst}\\
\text{)}
\end{proof}
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 \((\text{snd} \, P \, \sigma)\)
    thus \(?thesis \) using \(\text{trm-simp}(8)\) 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 \( \Gamma \vdash M : \tau \) \( y \not\in \text{fvs} \, M \)
shows \( \Gamma(y \mapsto \sigma) \vdash M : \tau \)
using assms proof (induction rule: typing.induct)
  case \( (\text{tunit} \, \Gamma) \)
  thus \(?case \) using \(\text{typing} \, . \text{tunit} \) by metis
next
  case \( (\text{tvar} \, \Gamma \, x \, \tau \, \sigma) \)
  hence \( y \neq x \) using \(\text{fvs-simp}(2)\) \, singletonI by force
  hence \( (\Gamma(y \mapsto \sigma)) \, x = \text{Some} \, \tau \) using \(\text{tvar} \, . \text{hyps} \, \text{fun-upd-apply} \) by simp
  thus \(?case \) using \(\text{typing} \, . \text{tvar} \) by metis
next
  case \( (\text{tapp} \, \Gamma \, f \, \tau \, \tau' \, x) \)
  from \( \langle y \not\in \text{fvs} \, (\text{App} \, f \, x) \rangle \) have \( y \not\in \text{fvs} \, f \, y \not\in \text{fvs} \, x \) using \(\text{fvs-simp}(3)\) \, Un-iff
by force+
  hence \( (\Gamma(y \mapsto \sigma)) \vdash f : (\text{TArr} \, \tau \, \tau') \) \( (\Gamma(y \mapsto \sigma)) \vdash x : \tau \) using \(\text{tapp} \, . \text{IH} \) by metis+
  thus \(?case \) using \(\text{typing} \, . \text{tapp} \) by metis
next
  case \( (\text{tfn} \, \Gamma \, x \, \tau \, A \, \tau') \)
  from \( \langle y \not\in \text{fvs} \, (\text{Fn} \, x \, \tau \, A) \rangle \) consider \( y = x \mid y \neq x \land y \not\in \text{fvs} \, A \)
using \(\text{fvs-simp}(4)\) \, 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 \(\text{tfn} \, . \text{hyps} \, \text{fun-upd-upd} \) by simp
    thus \(?thesis \) using \(\text{typing} \, . \text{tfn} \) by metis
next
    case 2
    hence \( y \neq x \) \( y \not\in \text{fvs} \, A \) by auto
hence \( (\Gamma(x \mapsto \tau, \, y \mapsto \sigma)) \vdash A : \tau' \) using \(\text{tfn} \, . \text{IH} \) by metis
hence \( (\Gamma(y \mapsto \sigma, \, x \mapsto \tau)) \vdash A : \tau' \) using \( y \neq x \) \, \text{fun-upd-twist} by metis

45
thus \thesis using \texttt{typing.tfn} by \texttt{metis}

next

\texttt{qed}

next
case (\texttt{tpair} Γ A τ B σ)
  thus ?case using \texttt{typing.tpair fvs-simp(5)} UnCI by \texttt{metis}

next
case (\texttt{tfst} Γ P τ σ)
  thus ?case using \texttt{typing.tfst fvs-simp(6)} by \texttt{metis}

next
case (\texttt{tsnd} Γ P τ σ)
  thus ?case using \texttt{typing.tsnd fvs-simp(7)} by \texttt{metis}

\texttt{qed}

\texttt{lift-definition} \texttt{infer :: 'a typing-ctx ⇒ 'a trm ⇒ type option is ptrm-infer-type using ptrm-infer-type-alpha-equiv.}

\texttt{export-code} \texttt{infer fresh-nat-inst.fresh-in-nat in Haskell}

\texttt{lemma} \texttt{infer-simp:}
\texttt{shows}
infer Γ Unit = Some TUnit
infer Γ (\texttt{Var} x) = Γ x
infer Γ (\texttt{App} A B) = (case (infer Γ A, infer Γ B) of
  (Some (\texttt{TArr} τ σ), Some τ') ⇒ (if τ = τ' then Some σ else None)
| - ⇒ None
)
infer Γ (\texttt{Fn} x τ A) = (case infer (Γ(x ⇒→ τ)) A of
  Some σ ⇒ Some (\texttt{TArr} τ σ)
| None ⇒ None
)
infer Γ (\texttt{Pair} A B) = (case (infer Γ A, infer Γ B) of
  (Some τ, Some σ) ⇒ Some (\texttt{TPair} τ σ)
| - ⇒ None
)
infer Γ (\texttt{Fst} P) = (case infer Γ P of
  (Some (\texttt{TPair} τ σ)) ⇒ Some τ
| - ⇒ None
)
infer Γ (\texttt{Snd} P) = (case infer Γ P of
  (Some (\texttt{TPair} τ σ)) ⇒ Some σ
| - ⇒ None
)
\texttt{by}((\texttt{transfer, simp}+) )

\texttt{lemma} \texttt{infer-unitE:}
\texttt{assumes} infer Γ Unit = Some τ
shows \( \tau = \text{TUnit} \)
using assms by (transfer, simp)

lemma infer-varE:
  assumes \( \text{infer } \Gamma \ (\text{Var } x) = \text{Some } \tau \)
  shows \( \Gamma \ x = \text{Some } \tau \)
using assms proof (transfer)
fix \( \Gamma \) :: 'a typing-ctx and \( A B \)
assume \( H \): ptrm-infer-type \( \Gamma \ (\text{PApp } A B) = \text{Some } \tau \)
have \( \text{ptrm-infer-type } \Gamma \ A = \text{None} \)
proof (rule classical, auto)
  fix \( x \)
  assume \( T = \text{TVar } x \)
  hence \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } (\text{TVar } x) \) using * by metis
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PApp } A B) = \text{None} \) by simp
  thus False using \( H \) by auto
qed
from this obtain \( T \) where *: \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } T \) by auto
have \( T \neq \text{TVar } x \) for \( x \)
proof (rule classical, auto)
  fix \( x \)
  assume \( T = \text{TVar } x \)
  hence \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } (\text{TVar } x) \) using * by metis
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PApp } A B) = \text{None} \) by simp
  thus False using \( H \) by auto
qed
moreover have \( T \neq \text{TUnit} \)
proof (rule classical, auto)
  fix \( x \)
  assume \( T = \text{TUnit} \)
  hence \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } \text{TUnit} \) using * by metis
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PApp } A B) = \text{None} \) by simp
  thus False using \( H \) by auto
qed
moreover have \( T \neq \text{TPair } \tau \sigma \) for \( \tau \sigma \)
proof (rule classical, auto)
  fix \( \tau \sigma \)
  assume \( T = \text{TPair } \tau \sigma \)
  hence \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } (\text{TPair } \tau \sigma) \) using * by metis
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PApp } A B) = \text{None} \) by simp
  thus False using \( H \) by auto
qed
ultimately obtain \( \sigma \tau' \) where \( T = \text{TArr } \sigma \tau' \) by (cases \( T \), blast, auto)
hence *: \( \text{ptrm-infer-type } \Gamma \ A = \text{Some } (\text{TArr } \sigma \tau') \) using * by metis

47
have \( \text{ptrm-infer-type } \Gamma \ B \neq \text{None} \)
proof\(\text{rule classical, auto} \)
  assume \( \text{ptrm-infer-type } \Gamma \ B = \text{None} \)
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PApp } A \ B) = \text{None} \) using * by auto
  thus False using \( H \) by auto
qed

from this obtain \( \sigma' \) where **: \( \text{ptrm-infer-type } \Gamma \ B = \text{Some } \sigma' \) by auto

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

hence **: \( \text{ptrm-infer-type } \Gamma \ B = \text{Some } \sigma \) using ** by auto

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

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

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

have \( \text{ptrm-infer-type } (\Gamma(\imath \mapsto T)) \ A \neq \text{None} \)
proof\(\text{rule classical, auto} \)
  assume \( \text{ptrm-infer-type } (\Gamma(\imath \mapsto T)) \ A = \text{None} \)
  hence \( \text{ptrm-infer-type } \Gamma \ (\text{PFN } x \ T \ A) = \text{None} \) by auto
  thus False using \( H \) by auto
qed

from this obtain \( \sigma \) where *: \( \text{ptrm-infer-type } (\Gamma(\imath \mapsto T)) \ A = \text{Some } \sigma \) by auto

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

lemma infer-pairE:
\begin{verbatim}
assumes infer \( \Gamma (\text{Pair} \ A \ B) = \text{Some} \ \tau \)
shows \( \exists T \ S. \ \tau = \text{TPair} \ T \ S \land \text{infer} \ \Gamma A = \text{Some} \ T \land \text{infer} \ \Gamma B = \text{Some} \ S \)
using assms proof(transfer)
fix \( A \ B :: \text{'a ptrm} \) and \( \Gamma \ \tau \)
assume \( H: \text{ptrm-infer-type} \ \Gamma (\text{PPair} \ A \ B) = \text{Some} \ \tau \)

have \( \text{ptrm-infer-type} \ \Gamma A \neq \text{None} \)
proof(rule classical, auto)
  assume \( \text{ptrm-infer-type} \ \Gamma A = \text{None} \)
  hence \( \text{ptrm-infer-type} \ \Gamma (\text{PPair} \ A \ B) = \text{None} \) by auto
  thus \( \text{False} \) using \( H \) by auto
qed

moreover have \( \text{ptrm-infer-type} \ \Gamma B \neq \text{None} \)
proof(rule classical, auto)
  assume \( \text{ptrm-infer-type} \ \Gamma B = \text{None} \)
  hence \( \text{ptrm-infer-type} \ \Gamma (\text{PPair} \ A \ B) = \text{None} \) by (simp add: option.case-eq-if)
  thus \( \text{False} \) using \( H \) by auto
qed

ultimately obtain \( T \ S \)
where \( \tau = \text{TPair} \ T \ S \text{\ text{ptrm-infer-type} \ \Gamma A = \text{Some} \ T \text{\ text{ptrm-infer-type} \ \Gamma B = \text{Some} \ S} \)
using \( H \) by auto
thus \( \exists T \ S. \ \tau = \text{TPair} \ T \ S \land \text{ptrm-infer-type} \ \Gamma A = \text{Some} \ T \land \text{ptrm-infer-type} \ \Gamma B = \text{Some} \ S \) by auto
qed

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

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

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

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

moreover have \( \text{ptrm-infer-type} \ \Gamma P \neq \text{Some} \ (\text{TArr} \ T \ S) \) for \( T \ S \)
\end{verbatim}
proof\text{(rule classical, auto)}
    \begin{align*}
    \text{assume } & \text{ptrm-infer-type } \Gamma P = \text{Some } (TArr T S) \\
    \text{thus } & \text{False using } H \text{ by simp}
    \end{align*}
qend
ultimately obtain \( T S \) where
    \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S)
    \text{using } type.d\text{istinct type} \circ\text{haust option} \circ\text{haust by metis}
moreover hence \text{ptrm-infer-type } \Gamma (PFst P) = \text{Some } T \text{ by simp}
ultimately show \( \exists T S. \) \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S) \land \tau = T
    \text{using } H \text{ by auto}
qend

\begin{lemma}
\text{infer-sndE:}
\begin{align*}
\text{assumes } & \text{infer } \Gamma (Snd P) = \text{Some } \tau \\
\text{shows } & \exists T S. \text{infer } \Gamma P = \text{Some } (TPair T S) \land \tau = S \\
\text{using } & \text{assms proof}(\text{transfer}) \\
\text{fix } & P :: \langle a \rangle \text{ ptrm and } \Gamma \tau \\
\text{assume } & H: \text{ptrm-infer-type } \Gamma (P\text{snd} P) = \text{Some } \tau
    \end{align*}
\begin{proof} (rule classical, auto)
    \begin{align*}
    \text{assume } & \text{ptrm-infer-type } \Gamma P = \text{None} \\
    \text{thus } & \text{False using } H \text{ by simp}
    \end{align*}
qend
moreover have \text{ptrm-infer-type } \Gamma P \neq \text{Some } TUnit
    \text{proof (rule classical, auto)}
    \begin{align*}
    \text{assume } & \text{ptrm-infer-type } \Gamma P = \text{Some } TUnit \\
    \text{thus } & \text{False using } H \text{ by simp}
    \end{align*}
qend
moreover have \text{ptrm-infer-type } \Gamma P \neq \text{Some } (TVar x) \text{ for } x
    \text{proof (rule classical, auto)}
    \begin{align*}
    \text{assume } & \text{ptrm-infer-type } \Gamma P = \text{Some } (TVar x) \\
    \text{thus } & \text{False using } H \text{ by simp}
    \end{align*}
qend
moreover have \text{ptrm-infer-type } \Gamma P \neq \text{Some } (TArr T S) \text{ for } T S
    \text{proof (rule classical, auto)}
    \begin{align*}
    \text{assume } & \text{ptrm-infer-type } \Gamma P = \text{Some } (TArr T S) \\
    \text{thus } & \text{False using } H \text{ by simp}
    \end{align*}
qend
ultimately obtain \( T S \) where
    \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S)
    \text{using } type.d\text{istinct type} \circ\text{haust option} \circ\text{haust by metis}
moreover hence \text{ptrm-infer-type } \Gamma (P\text{fst} P) = \text{Some } S \text{ by simp}
ultimately show \( \exists T S. \) \text{ptrm-infer-type } \Gamma P = \text{Some } (TPair T S) \land \tau = S
    \text{using } H \text{ by auto}
qend
\end{lemma}

\begin{lemma}
\text{infer-sound:}
\begin{align*}
\text{assumes } & \text{infer } \Gamma M = \text{Some } \tau
    \end{align*}
\end{lemma}
\[ \Gamma \vdash M : \tau \]

Using assms proof:

\( \text{induction } M \text{ arbitrary; } \Gamma \tau \text{ rule: trm-induct} \)

Case 1

Thus \( \text{?case using infer-unitE typing.unit by metis} \)

Next

Case \( (2 \, x) \)

Hence \( \Gamma \, x = \text{Some } \tau \text{ using infer-varE by metis} \)

Thus \( \text{?case using typing.tvar by metis} \)

Next

Case \( (3 \, A \, B) \)

From \( \text{infer } \Gamma \, (\text{App } A \, B) = \text{Some } \tau \text{ obtain } \sigma \)

Where \( \text{infer } \Gamma \, A = \text{Some } (\text{TArr } \sigma \, \tau) \text{ and infer } \Gamma \, B = \text{Some } \sigma \)

Using \( \text{infer-appE by metis} \)

Thus \( \text{?case using 3.IH typing.tapp by metis} \)

Next

Case \( (4 \, x \, T \, A \, \Gamma \, \tau) \)

From \( \text{infer } \Gamma \, (\text{Fn } x \, T \, A) = \text{Some } \tau \text{ obtain } \sigma \)

Where \( \tau = \text{TArr } T \, \sigma \text{ and infer } (\Gamma(x \mapsto T)) \, A = \text{Some } \sigma \)

Using \( \text{infer-fnE by metis} \)

Thus \( \text{?case using 4.IH typing.tfn by metis} \)

Next

Case \( (5 \, A \, B \, \Gamma \, \tau) \)

Thus \( \text{?case using typing.tpair infer-pairE by metis} \)

Next

Case \( (6 \, P \, \Gamma \, \tau) \)

Thus \( \text{?case using typing.tfst infer-fstE by metis} \)

Next

Case \( (7 \, P \, \Gamma \, \tau) \)

Thus \( \text{?case using typing.tsnd infer-sndE by metis} \)

Next

Qed

Lemma \( \text{infer-complete} : \)

Assumes \( \Gamma \vdash M : \tau \)

Shows \( \text{infer } \Gamma \, M = \text{Some } \tau \)

Using assms proof:

\( \text{induction} \)

Case \( (\text{tfn } \Gamma \, x \, \tau \, A \, \sigma) \)

Thus \( \text{?case by (simp add: infer-simp(4) tfn.IH)} \)

Next

Qed (auto simp add: infer-simp)

Theorem \( \text{infer-valid} : \)

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: \( \text{x = y} \) \( \Rightarrow \) substitutes (Var x) y M M
| var2: \( \text{x \neq y} \) \( \Rightarrow \) substitutes (Var x) y M (Var x)
lemma substitutes-prm:
  assumes substitutes A x M A'
  shows substitutes (π · A) (π $ x) (π · M) (π · 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 fvs A' ⊆ fvs A = {x} ∪ 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

52
next

case (var2 \( x \) \( y \) \( M \))
  thus ?case
    using fus-simp(2) Un-subset-iff Un-upper2 insert-Diff-if insert-is-Un single-tonD sup-commute
    by metis

next

case (app \( A \) \( x \) \( M \) \( A' \) \( B' \))
  hence \( \text{fvs} \ A' \cup \text{fvs} \ B' \subseteq (\text{fvs} \ A - \{x\} \cup \text{fvs} \ M) \cup (\text{fvs} \ B - \{x\} \cup \text{fvs} \ M) \) by auto
  hence \( \text{fvs} \ A' \cup \text{fvs} \ B' \subseteq (\text{fvs} \ A \cup \text{fvs} \ B) - \{x\} \cup \text{fvs} \ M \) by auto
  thus ?case using fus-simp(3) by metis

next

case (fn \( x \) \( y \) \( M \) \( A A' T \))
  hence \( \text{fvs} \ A' - \{y\} \subseteq \text{fvs} \ A - \{y\} - \{x\} \cup \text{fvs} \ M \) by auto
  thus ?case using fus-simp(4) by metis

next

case (pair \( A \) \( x \) \( M \) \( A' \) \( B' \))
  hence \( \text{fvs} \ A' \cup \text{fvs} \ B' \subseteq (\text{fvs} \ A - \{x\} \cup \text{fvs} \ M) \cup (\text{fvs} \ B - \{x\} \cup \text{fvs} \ M) \) by auto
  hence \( \text{fvs} \ A' \cup \text{fvs} \ B' \subseteq (\text{fvs} \ A \cup \text{fvs} \ B) - \{x\} \cup \text{fvs} \ M \) by auto
  thus ?case using fus-simp(5) by metis

next

case (fst \( P \) \( x \) \( M \) \( P' \))
  thus ?case using fus-simp(6) by fastforce

next

case (snd \( P \) \( x \) \( M \) \( P' \))
  thus ?case using fus-simp(7) by fastforce

next

dqed

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 \( y = g \) 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 \( \text{Var} \ x \) \( y \) \( M \) \( X \)

lemma substitutes-varE:
  assumes substitutes \( \text{Var} \ x \) \( y \) \( M \) \( X \)
  shows \( (x = y \land M = X) \lor (x \neq y \land X = \text{Var} \ x) \)
  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   ∃ A' B'. substitutes A x M A' ∧ substitutes B x M B' ∧ X = 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 /= x y /∈ fvs M
shows   ∃ A'. substitutes A x M A' ∧ X = 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 ∧ T = S ∧ A = B | y /= z ∧ T = S ∧ y /∈ fvs B ∧ A = [y ↔ z] · B
using (Fn y T A = Fn z S B) trm-simp(4) by metis
thus ?case proof(cases)
case 1
thus ?thesis using 6 by metis
next
case 2
hence y /= z T = S y /∈ fvs B A = [y ↔ z] · B by auto
have substitutes ([y ↔ z] · B) ([y ↔ z] $ x) ([y ↔ z] · M) ([y ↔ z] · B')
using substitutes-prm (substitutes B x M B'') by metis
hence substitutes A ([y ↔ z] $ x) ([y ↔ z] · M) ([y ↔ z] · B')
using 'A = [y ↔ z] · B' by metis
hence substitutes A x ([y ↔ z] · M) ([y ↔ z] · B')
using ⟨y /= x | x /= z⟩ prm-unit-inaction by metis
hence *: substitutes A x M ([y ↔ z] · B')
using ⟨y /= fvs M | z /= fvs M⟩ trm-prm-unit-inaction by metis
have y /= fvs B'
using substitutes-fvs (substitutes B x M B'' | y /= fvs B' | y /= fvs M)
**Diff-subset UnE rev-subsetD**

by blast

**hence** \( X = \text{Fn} \ y \ T (\{ y \leftrightarrow z \} \cdot B') \)

**using** \( X = \text{Fn} \ z \ S \ B' \land y \neq z \land T = S \text{ fn-eq} \)

by metis

**thus** \(?thesis** using * by auto**

next

qed

next

qed ( )

**inductive-cases substitutes-pairE': substitutes \((\text{Pair} A B) x M X\)**

**lemma substitutes-pairE:**

**assumes** substitutes \((\text{Pair} A B) x M X\)

**shows** \( \exists A' B'. \text{substitutes} A x M A' \land \text{substitutes} B x M B' \land X = \text{Pair} A' B' \)

**proof** (cases rule: substitutes-pairE'[\text{where} \ A=A \text{ and} \ B=B \text{ and} \ x=x \text{ and} \ M=M \text{ and} \ X=X])

**case** \((7 A A' B B')\)

**thus** \(?thesis** using **trm-simp(5) trm-simp(6)** by blast

next

qed ( )

**inductive-cases substitutes-fstE': substitutes \((\text{Fst} P) x M X\)**

**lemma substitutes-fstE:**

**assumes** substitutes \((\text{Fst} P) x M X\)

**shows** \( \exists P'. \text{substitutes} P x M P' \land X = \text{Fst} P' \)

**proof** (cases rule: substitutes-fstE'[\text{where} \ P=P \text{ and} \ x=x \text{ and} \ M=M \text{ 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-fst  
)

inductive-cases substitutes-sndE':: substitutes (Snd P) x M X

lemma substitutes-sndE:  
assumes "∃ P'. substitutes P x M P' ∧ X = Snd P'
shows (cases rule: substitutes-sndE[where P=P' and x=x and M=M and X=X])
proof (cases rule: substitutes-sndE[where P=P' and x=x and M=M and X=X])
  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 "∃ X. substitutes A x M X
proof (induction A rule: trm-strong-induct[where S={x} ∪ fvs M])
show finite ({x} ∪ fvs M) using fes-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 ≠ 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


56
obtain $X$ where $X = (\text{Var } y)$ by auto

hence substitutes $\langle x \neq y \rangle$ substitutes.var2 by metis

thus $\text{thesis}$ by auto

next

qed

next
case $(\exists A B)$

from this obtain $A' B'$ where $A'$: substitutes $A \times M A'$ and $B'$: substitutes $B \times M B'$ by auto

obtain $X$ where $X = \text{App } A' B'$ by auto

hence substitutes $\langle \text{App } A B \rangle x M X$ using $A' B'$ substitutes.app by metis

thus $\?case$ by auto

next
case $(\exists y T A)$

from this obtain $A'$ where $A'$: substitutes $A \times M A'$ by auto

from $y \notin (\{x\} \cup \text{fvs } M)$ have $y \neq x \land y \notin \text{fvs } M$ by auto

obtain $X$ where $X = \text{Fn } y T A'$ by auto

hence substitutes $\langle \text{Fn } y T A \rangle x M X$ using substitutes.fn $\langle y \neq x \rangle \langle y \notin \text{fvs } M \rangle$ $A'$ by metis

thus $\?case$ by auto

next
case $(\exists 5 A B)$

from this obtain $A' B'$ where substitutes $A \times M A'$ substitutes $B \times M B'$ by auto

from this obtain $X$ where $X = \text{Pair } A' B'$ by auto

hence substitutes $\langle \text{Pair } A B \rangle x M X$ using substitutes.pair $\langle \text{substitutes } A \times M A' \rangle \langle \text{substitutes } B \times M B' \rangle$

by metis

thus $\?case$ by auto

next
case $(\exists 6 P)$

from this obtain $P'$ where substitutes $P \times M P'$ by auto

from this obtain $X$ where $X = \text{Fst } P'$ by auto

hence substitutes $\langle \text{Fst } P \rangle x M X$ using substitutes.fst $\langle \text{substitutes } P \times M P' \rangle$

by metis

thus $\?case$ by auto

next
case $(\exists 7 P)$

from this obtain $P'$ where substitutes $P \times M P'$ by auto

from this obtain $X$ where $X = \text{Snd } P'$ by auto

hence substitutes $\langle \text{Snd } P \rangle x M X$ using substitutes.snd $\langle \text{substitutes } P \times M P' \rangle$

by metis

thus $\?case$ by auto

next

qed

lemma substitutes-unique:

assumes substitutes $A \times M B$ substitutes $A \times M C$

shows $B = C$
using assms proof (induction A arbitrary: B C rule: trm-strong-induct [where
S=\{x\} \cup \text{fvs} M])
  show finite (\{x\} \cup \text{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 \neq x and y \notin \text{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 :: \('a trm \Rightarrow 'a \Rightarrow 'a trm\) \(-\cdot ::= \cdot\) where
subst A x M \equiv (THE X. substitutes A x M X)

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

lemma subst-simp-var1:
  shows (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 (Var x)[y ::= M] = 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[x ::= M]) and 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 ≠ y y /∈ fvs M
  shows (Fn y T A)[x ::= M] = Fn y T (A[x ::= M])
unfolding subst-def proof
  obtain A' where 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[x ::= M]) and 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 \( \ast \): 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 \( \ast \) 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[x ::= M]) by auto
hence substitutes P x M P' unfolding subst-def using substitutes-function the1
by metis
hence substitutes (Fst P) x M (Fst P') using substitutes.fst by metis
thus \( \ast \): 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 \( \ast \) 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[x ::= M]) by auto
hence substitutes P x M P' unfolding subst-def using substitutes-function the1
by metis
hence substitutes (Snd P) x M (Snd P') using substitutes.snd by metis
thus \( \ast \): 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 \( \ast \) 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-fes 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 fes-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 \setminus \{x\} using \{y \neq x\} \notin \text{fvs} (\text{Fn} x T A) fvs-simp(4) by metis
  hence y \notin \text{fvs} A using \{y \neq x\} by auto
  hence A[y := N] = A using 4.IH by blast
  thus ?case using \{y \neq x\} \notin \text{fvs} A \setminus \{x\} \notin \text{fvs} N 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 fes-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 $\triangleleft$case proof\{cases\}
  case 1
    thus $\triangleleft$thesis using $subst-simp-var1$ $trm-prm-simp(2)$ $prm-unit-action$
    $prm-unit-commutes$ by metis
    next
    case 2
      thus $\triangleleft$thesis using $subst-simp-var2$ $trm-prm-simp(2)$ $prm-unit-inaction$ $:\ y
      \neq x)$ by metis
      qed
    next
  case $(\exists A B)$
    from $(y \notin fvs (App A B))$ have $y \notin fvs A \ y \notin fvs B$ by (auto simp add:
    $fvs-simp(3)$)
    thus $\triangleleft$case using $3.IH$ $subst-simp-app$ $trm-prm-simp(3)$ by metis
    next
  case $(\lambda x \ T \ A)$
    hence $x \neq y \ x \neq z \ x \notin fvs M$ by auto
    hence $\ast: (\langle y \leftrightarrow z \rangle :: A)[y :: M] = (A[z :: M])$ using $4.IH$ by metis
    have $(\langle y \leftrightarrow z \rangle \cdot (\langle y \leftrightarrow z \rangle \cdot A))[y :: M] = ((\langle y \leftrightarrow z \rangle \cdot (\langle y \leftrightarrow z \rangle \cdot A))[y :: M])$
      using $trm-prm-simp(4)$ by metis
    also have $\ldots = ((\langle y \leftrightarrow z \rangle \cdot (\langle y \leftrightarrow z \rangle \cdot A))[y :: M])$
      using $prm-unit-inaction$ $(x \neq y) \ (x \neq z)$ by metis
    also have $\ldots = (\langle y \leftrightarrow z \rangle \cdot (\langle y \leftrightarrow z \rangle \cdot A))[y :: M])$
      using $subst-simp-fn$ $(x \neq y) \ (x \notin fvs M)$ by metis
    also have $\ldots = (\langle y \leftrightarrow z \rangle \cdot (\langle y \leftrightarrow z \rangle \cdot A))[y :: M])$
      using $subst-simp-fn$ $(x \neq z) \ (x \notin fvs M)$ by metis
    finally show $\triangleleft$case.
  next
  case $(\lambda A B)$
    from $(y \notin fvs (Pair A B))$ have $y \notin fvs A \ y \notin fvs B$ by (auto simp add:
    $fvs-simp(5)$)
    hence $(\langle y \leftrightarrow z \rangle \cdot A)[y :: M] = (A[z :: M]) (\langle y \leftrightarrow z \rangle \cdot B)[y :: M] = (B[z :: M])$
      using $5.IH$ by metis
    thus $\triangleleft$case using $trm-prm-simp(5)$ $subst-simp-pair$ by metis
  next
  case $(\lambda P)$
    from $(y \notin fvs (Fst P))$ have $y \notin fvs P$ by (simp add: $fvs-simp(6)$)
    hence $(\langle y \leftrightarrow z \rangle \cdot P)[y :: M] = (P[z :: M])$ using $6.IH$ by metis
    thus $\triangleleft$case using $trm-prm-simp(6)$ $subst-simp-fst$ by metis
  next
  case $(\lambda P)$
from ⟨\notin \text{fvs} (\text{snd } P)⟩ have \(y \notin \text{fvs } P\) by (simp add: fvs-simp(7))

hence \((y \leftrightarrow z) : P[y := M] = (P[z := M])\) using 7.IH by metis

thus \(?\text{case using } \text{trm-prm-simp(7)} \text{ subst-simp-sn}\) by metis

next

qed

lemma barendregt:

assumes \(y \neq z\) \(y \notin \text{fvs } L\)

shows \(M[y := N][z := L] = (M[z := L][y := N[z := L]])\)

using \(\text{assms}\)

proof (induction \(M\) rule: \(\text{trm-strong-induct}\) where \(S = \{y, z\} \cup \text{fvs } N \cup \text{fvs } L\))

show finite \((\{y, z\} \cup \text{fvs } N \cup \text{fvs } L)\) using \(\text{fvs-finite}\) by auto

next case 1

thus \(?\text{case using } \text{subst-simp-unit by metis}\)

next

case \((2 \ x)\)

consider \(x = y \mid x = z \mid x \neq y \land x \neq z\) by auto

thus \(?\text{case proof(cases)}\)

  case 1

  hence \(x = y \land x \neq z\) using \(\text{assms(1)}\) by auto

  thus \(?\text{thesis using } \text{subst-simp-var1 subst-simp-var2 by metis}\)

next

case 2

  hence \(x \neq y \land x = z\) using \(\text{assms(1)}\) by auto

  thus \(?\text{thesis using } \text{subst-free } y \notin \text{fvs } L \text{ subst-simp-var1 subst-simp-var2 by metis}\)

by metis

next

case 3

  then show \(?\text{thesis using } \text{subst-simp-var2 by metis}\)

next

next

case \((3 \ A \ B)\)

thus \(?\text{case using } \text{subst-simp-app by metis}\)

next

case \((4 \ x \ T \ A)\)


from \(x \notin \{y, z\} \cup \text{fvs } N \cup \text{fvs } L\) have \(x \neq y \land x \neq z \land x \notin \text{fvs } N \land x \notin \text{fvs } L\) by auto

hence \(x \notin \text{fvs } (N[z := L])\) using \(\text{subst-fes by auto}\)

have \((\text{Fn } x \ T \ A)[y := N][z := L] = \text{Fn } x \ T \ (A[y := N][z := L])\) using \(\text{subst-fs by auto}\)

also have \(\ldots = \text{Fn } x \ T \ (A[z := L][y := N[z := L]])\) using \(* \text{ by metis}\)

also have \(\ldots = ((\text{Fn } x \ T \ A)[z := L][y := N[z := L]])\) using \(\text{subst-fs by auto}\)

by metis

next

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

lemma \(\text{typing-subst}:\)
\(\text{assumes } \Gamma(x \mapsto \tau) \vdash M : \sigma \Gamma \vdash N : \tau\)
\(\text{shows } \Gamma \vdash M[z := N] : \sigma\)
\(\text{using assms proof(induction } M \text{ arbitrary;} \Gamma \sigma \text{ rule: \text{trm-strong-depth-induct}}[\text{where } S = \{z\} \cup \text{fvs } N])\)
\(\text{show finite } (\{z\} \cup \text{fvs } N) \text{ using fvs-finite by auto}\)
next
case 1
  \(\text{thus } ?case \text{ using subst-simp-unit typing \ tunit typing-unitE by metis}\)
next
case (2 x)
  \(\text{hence } *: (\Gamma(x \mapsto \tau)) \ x = \text{Some } \sigma \text{ using typing-varE by metis}\)
  consider \(x = z \mid x \neq z\) by auto
  \(\text{thus } ?case \text{ proof(cases)}\)
  case 1
    \(\text{hence } (\Gamma(x \mapsto \tau)) \ x = \text{Some } \sigma \text{ using } * \text{ by metis}\)
    \(\text{hence } \tau = \sigma \text{ by auto}\)
    \(\text{thus } ?thesis \text{ using } (\Gamma \vdash N : \tau) \text{ subst-simp-var1 } (x = z) \text{ by metis}\)
  next
case 2
    \(\text{hence } \Gamma \ x = \text{Some } \sigma \text{ using } * \text{ by auto}\)
    \(\text{hence } \Gamma \vdash \text{Var } x : \sigma \text{ using typing-tvar by metis}\)
    \(\text{thus } ?thesis \text{ using } (x \neq z) \text{ subst-simp-var2 by metis}\)
  next
next
\(\text{next}\)
case (3 A B)
\(\text{have } *: \text{depth } A < \text{depth } (\text{App } A \ B) \land \text{depth } B < \text{depth } (\text{App } A \ B)\)
\(\text{using depth-app by auto}\)
\(\text{from } \Gamma(z \mapsto \tau) \vdash \text{App } A \ B : \sigma; \text{ obtain } \sigma’ \text{ where }\)
\(\Gamma(z \mapsto \tau) \vdash A : (\text{TArr } \sigma’ \sigma)\)
\(\Gamma(z \mapsto \tau) \vdash B : \sigma’\)
\(\text{using typing-appE by metis}\)
hence
\[ \Gamma \vdash (A[z := N]) : (T\text{Arr} \sigma' \sigma) \]
\[ \Gamma \vdash (B[z := N]) : \sigma' \]
using 3 * by metis+

hence \[ \Gamma \vdash \text{App} (A[z := N]) (B[z := N]) : \sigma \text{ using typing.tapp by metis} \]

thus ?case using subst-simp-app by metis
next
case (4 x T A)
hence \[ x \neq z \text{ x } \notin \text{ fvs } N \text{ by auto} \]
hence \[ * : \Gamma(x \Rightarrow T) \vdash N : \tau \text{ using typing-weaken 4 by metis} \]

have **: depth A < depth (Fn x T A) using depth-fn.

from \[ \Gamma(z \Rightarrow \tau) \vdash \text{Fn x T A} : \sigma \text{ obtain } \sigma' \text{ where} \]
\[ \sigma = T\text{Arr} T \sigma' \]
\[ \Gamma(z \Rightarrow \tau)(x \Rightarrow T) \vdash A : \sigma' \]

using typing-fnE by metis

hence \[ \Gamma(x \Rightarrow T)(z \Rightarrow \tau) \vdash A : \sigma' \text{ using } (x \neq z) \text{ fun-upd-twist by metis} \]

hence \[ \Gamma(x \Rightarrow T) \vdash A[z := N] : \sigma' \text{ using } 4 * ** \text{ by metis} \]

hence \[ \Gamma \vdash \text{Fn x T A}[z := N] : \sigma \text{ using typing.tfn } \sigma = T\text{Arr} T \sigma' \text{ by metis} \]

thus ?case using \( (x \neq z) \cdot (x \notin \text{ fvs } N) \cdot \text{ subst-simp-fn by metis} \)
next
case (5 A B)
from this obtain \( T S \) where \( \sigma = \text{TPair} T S \)
\[ \Gamma(z \Rightarrow \tau) \vdash A : T \Gamma(z \Rightarrow \tau) \vdash B : S \]

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 \[ \Gamma \vdash A[z := N] : T \Gamma \vdash B[z := N] : S \text{ using 5 by metis+} \]

hence \[ \Gamma \vdash \text{Pair} (A[z := N]) (B[z := N]) : \sigma \text{ using } \langle \sigma = T\text{Pair} T S \rangle \text{ typing.tpair by metis} \]

thus ?case using subst-simp-pair by metis
next
case (6 P)
from this obtain \( \sigma' \) where \( \Gamma(z \Rightarrow \tau) \vdash P : (\text{TPair } \sigma \sigma') \text{ using typing-fstE by metis} \)

moreover have depth P < depth (Fst P) using depth-fst by metis

ultimately have \[ \Gamma \vdash P[z := N] : (\text{TPair } \sigma \sigma') \text{ using 6 by metis} \]

hence \[ \Gamma \vdash \text{Fst} (P[z := N]) : \sigma \text{ using } \text{typing.tfst by metis} \]

thus ?case using subst-simp-fst by metis
next
case (7 P)
from this obtain \( \sigma' \) where \( \Gamma(z \Rightarrow \tau) \vdash P : (\text{TPair } \sigma' \sigma) \text{ using typing-sndE by metis} \)

moreover have depth P < depth (Snd P) using depth-snd by metis

ultimately have \[ \Gamma \vdash P[z := N] : (\text{TPair } \sigma' \sigma) \text{ using 7 by metis} \]

hence \[ \Gamma \vdash \text{Snd} (P[z := N]) : \sigma \text{ using } \text{typing.tsnd by metis} \]

thus ?case using subst-simp-snd by metis
next

65
\begin{verbatim}
Qed

Inductive beta-reduction :: 'a trm ⇒ 'a trm ⇒ bool (-→β -) where
beta: (App (Fn x T A) M) →β A[x ::= M]
| app1: A →β A' →β (App A B) →β (App A' B)
| app2: B →β B' →β (App A B) →β (App A B')
| fn: A →β A' →β (Fn x T A) →β (Fn x T A')
| pair1: A →β A' →β (Pair A B) →β (Pair A' B)
| pair2: B →β B' →β (Pair A B) →β (Pair A B')
| fst1: P →β P' →β (Fst P) →β (Fst P')
| fst2: (Fst (Pair A B)) →β A
| snd1: P →β P' →β (Snd P) →β (Snd P')
| snd2: (Snd (Pair A B)) →β B

Lemma beta-reduction-fvs:
assumes M →β M'
shows fvs M' ⊆ 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' ∪ fvs B ⊆ fvs A ∪ fvs B by auto
    thus ?case using fvs-simp(3) by metis
next
  case (app2 A B A' B')
    hence fvs A ∪ fvs B' ⊆ fvs A ∪ fvs B by auto
    thus ?case using fvs-simp(3) by metis
next
  case (fn A A' x T)
    hence fvs A' - {x} ⊆ fvs A - {x} by auto
    thus ?case using fvs-simp(4) by metis
next
  case (pair1 A A' B)
    hence fvs A' ∪ fvs B ⊆ fvs A ∪ fvs B by auto
    thus ?case using fvs-simp(5) by metis
next
  case (pair2 A B A' B')
    hence fvs A ∪ fvs B' ⊆ fvs A ∪ 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')
\end{verbatim}
thus \textbf{case using} fvs-simp(7) \textbf{by} metis
next
\textbf{case} (snd2 A B)
\hspace*{1cm} thus \textbf{case using} fvs-simp(5, 7) \textbf{by} force
next
\textbf{qed}

\textbf{lemma} beta-reduction-prm:
\textbf{assumes} \( M \rightarrow_{\beta} M' \)
\textbf{shows} \( (\pi \cdot M) \rightarrow_{\beta} (\pi \cdot M') \)
\textbf{using} \textbf{assms} \textbf{by}(induction, auto simp add: beta-reduction.intro trm-prm-simp subst-prm)

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

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

\textbf{lemma} beta-reduction-left-appE:
\textbf{assumes} (App A B) \(\rightarrow_{\beta} M' \)
\textbf{shows}
\( (\exists x T X. A = (\text{Fn } x T X) \land M' = (X[x := B])) \lor \)
\( (\exists A'. (A \rightarrow_{\beta} A') \land M' = \text{App } A' B) \lor \)
\( (\exists B'. (B \rightarrow_{\beta} B') \land M' = \text{App } A B') \)
\textbf{using} \textbf{assms} \textbf{by}(
\hspace*{1cm} cases,
\hspace*{1cm} \text{metis trm-simp(2, 3),}
\hspace*{1cm} \text{metis trm-simp(2, 3),}
\hspace*{1cm} \text{metis trm-simp(2, 3),}
\hspace*{1cm} \text{auto simp add: app-not-fn app-not-pair app-not-fst app-not-snd}
\)

\textbf{lemma} beta-reduction-left-fnE:
\textbf{assumes} (Fn x T A) \(\rightarrow_{\beta} M' \)
\textbf{shows} \( \exists A'. (A \rightarrow_{\beta} A') \land M' = (\text{Fn } x T A') \)
\textbf{using} \textbf{assms} \textbf{proof}(cases)
\textbf{case} (fn B B' y S)
\hspace*{1cm} consider \( x = y \land T = S \land A = B \mid x \neq y \land T = S \land x \notin \text{fvs } B \land A = [x \leftrightarrow y] \cdot B \)
\hspace*{1cm} \textbf{using} trm-simp(4) (Fn x T A = Fn y S B) \textbf{by} metis
\hspace*{1cm} thus \textbf{?thesis} \textbf{proof}(cases)

67
case 1
  thus \( ?\text{thesis using fn by auto} \)
next
case 2
  thus \( ?\text{thesis using fn beta-reduction-fes beta-reduction-prm fn-eq by fastforce} \)
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 \( (\text{Pair } A \ B) \rightarrow \beta \ M' \)
  shows \( (\exists A'. (A \rightarrow \beta \ A') \land M' = (\text{Pair } A' \ B)) \lor (\exists B'. (B \rightarrow \beta \ B') \land M' = (\text{Pair } A \ B')) \)
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 \( (\text{Fst } P) \rightarrow \beta \ M' \)
  shows \( (\exists P'. (P \rightarrow \beta \ P') \land M' = (\text{Fst } P')) \lor (\exists A \ B. P = (\text{Pair } A \ B) \land M' = A) \)
using assms proof(cases)
  case (fst1 P P')
    thus \( ?\text{thesis using trm-simp(7) by blast} \)
next
  case (fst2 B)
    thus \( ?\text{thesis using trm-simp(7) by blast} \)
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

68
lemma beta-reduction-left-sndE:
  assumes \((\text{Snd } P) \rightarrow \beta M')\)
  shows \((\exists P'. (P \rightarrow \beta P') \land M' = (\text{Snd } P')) \lor (\exists A B. \text{P = Pair } A \ B \land M' = B)\)
  using assms proof (cases)
    case (snd1 P P')
    thus \(\text{thesis using trm-simp(8) by blast}\)
  next
    case (snd2 A)
    thus \(\text{thesis using trm-simp(8) by blast}\)
  next

lemma preservation':
  assumes \(\Gamma \vdash M : \tau \land M \rightarrow \beta M'\)
  shows \(\Gamma \vdash M' : \tau\)
  using assms proof (induction arbitrary: \(M'\) rule: typing.induct)
    case (tunit \(\Gamma\))
    thus \(\text{?case using beta-reduction-left-unitE by metis}\)
  next
    case (tvar \(\Gamma\) \(x\) \(\tau\))
    thus \(\text{?case using beta-reduction-left-varE by metis}\)
  next
    case (tapp \(\Gamma\) \(A\) \(\sigma\) \(B\) \(M'\))
    from \((\text{App } A \ B) \rightarrow \beta M'\) consider
    \((\exists x T X. A = (\text{Fn } x \ T \ X) \land M' = (X[x := B]))\) |
    \((\exists A'. (A \rightarrow \beta A') \land M' = \text{App } A' \ B)\) |
    \((\exists B'. (B \rightarrow \beta B') \land M' = \text{App } A \ B')\) using beta-reduction-left-appE by metis
    thus \(\text{?case proof(cases)\}\)
    case 1
    from this obtain \(x T X\) where \(A = \text{Fn } x \ T \ X\) and \(*: M' = (X[x := B])\)
    by auto
    have \(\Gamma(x \mapsto \tau) \vdash X : \sigma\) using typing-fnE \(\Gamma\vdash A : (T\text{Arr } \tau \ \sigma); \ A = \text{Fn } x \ T \ X\) type.inject
    by blast
    hence \(\Gamma \vdash (X[x := B]) : \sigma\) using typing-subst \(\Gamma \vdash B : \tau\) by metis

69
thus \textit{thesis} using \textit{*} by \textit{auto}

next

case 2

from this obtain \(A'\) where \(A \to \beta A'\) and \(*\): \(M' = (\text{App } A' \ B)\) by \textit{auto}

hence \(\Gamma \vdash A' : (\text{TArr } \tau \sigma)\) using \textit{tapp.IH}(1) by \textit{metis}

hence \(\Gamma \vdash (\text{App } A' \ B) : \sigma\) using \(\Gamma \vdash B : \tau\); \textit{typing.tapp} by \textit{metis}

thus \textit{thesis} using \textit{*} by \textit{auto}

next

case 3

from this obtain \(B'\) where \(B \to \beta B'\) and \(*\): \(M' = (\text{App } A \ B')\) by \textit{auto}

hence \(\Gamma \vdash (\text{App } A \ B') : \sigma\) using \(\Gamma \vdash A : (\text{TArr } \tau \sigma)\); \textit{typing.tapp} by \textit{metis}

thus \textit{thesis} using \textit{*} by \textit{auto}

next

qed

next

case (\textit{tfn} \(\Gamma \ x \ T \ A \ \sigma\))

from this obtain \(A'\) where \(A \to \beta A'\) and \(*\): \(M' = (\text{Fn } x \ T \ A')\)

using \textit{beta-reduction-left-fnE} by \textit{metis}

hence \(\Gamma(x \mapsto T) \vdash A' : \sigma\) using \textit{tfn.IH} by \textit{metis}

hence \(\Gamma \vdash (\text{Fn } x \ T \ A') : (\text{TArr } T \sigma)\) using \textit{typing.tfn} by \textit{metis}

thus \textit{case} using \textit{*} by \textit{auto}

next

case (\textit{tpair} \(\Gamma \ A \ \tau \ B \ \sigma\))

from this consider

\[ \exists A'. (A \to \beta A') \land M' = (\text{Pair } A' \ B) \]
\[ \lor \exists B'. (B \to \beta B') \land M' = (\text{Pair } A \ B') \]

using \textit{beta-reduction-left-pairE} by \textit{metis}

thus \textit{case proof}(cases)

case 1

from this obtain \(A'\) where \(A \to \beta A'\) and \(M' = \text{Pair } A' \ B\) by \textit{auto}

thus \textit{thesis} using \textit{tpair typing.tpair} by \textit{metis}

next

case 2

from this obtain \(B'\) where \(B \to \beta B'\) and \(M' = \text{Pair } A \ B'\) by \textit{auto}

thus \textit{thesis} using \textit{tpair typing.tpair} by \textit{metis}

next

next

next

case (\textit{tfst} \(\Gamma \ P \ \tau \ \sigma\))

from this consider

\[ \exists P'. (P \to \beta P') \land M' = \text{Fst } P' \]
\[ \lor \exists A \ B. P = \text{Pair } A \ B \land M' = A \text{ using } \textit{beta-reduction-left-fstE} \text{ by } \textit{metis} \]

thus \textit{case proof}(cases)

case 1

from this obtain \(P'\) where \(P \to \beta P'\) and \(M' = \text{Fst } P'\) by \textit{auto}

thus \textit{thesis} using \textit{tfst typing.tfst} by \textit{metis}

next

case 2

next

next

next

next

next

next

70
from this obtain \( A B \) where \( P = \text{Pair} \ A \ B \) and \( M' = A \) by auto
thus \(?\)thesis using \( \Gamma \vdash P : (T\text{Pair} \, \tau \, \sigma) \) typing-pairE by blast
next
qed
next
case (\text{tsnd} \ \Gamma \ P \ \tau \ \sigma)
from this consider
\[ \exists P'. \ (P \rightarrow \beta \ P') \land M' = \text{Snd} \ P' \]
| \[ \exists A \ B. \ P = \text{Pair} \ A \ B \land M' = B \] using beta-reduction-left-sndE by metis
thus \(?\)case proof(cases)
\begin{align*}
\text{case 1} & \quad \text{from this obtain } P' \text{ where } P \rightarrow \beta \ P' \text{ and } M' = \text{Snd} \ P' \text{ by auto} \\
& \quad \text{thus } ?\text{thesis using tsnd typing-tsnd by metis} \\
\text{next} \\
\text{case 2} & \quad \text{from this obtain } A B \text{ where } P = \text{Pair} \ A \ B \text{ and } M' = B \text{ by auto} \\
& \quad \text{thus } ?\text{thesis using } (\Gamma \vdash P : (T\text{Pair} \, \tau \, \sigma)) \text{ typing-pairE by blast} \\
\text{next} \\
\text{qed} \\
\text{next} \\
\text{qed}
\end{align*}

inductive \( NF :: 'a \text{ trm} \Rightarrow \text{bool} \) where
\begin{align*}
\text{unit: } & NF \ \text{Unit} \\
\text{var: } & NF \ (\text{Var} \ x) \\
\text{app: } & [A \neq Fn \ x \ T \ A'; \ NF \ A; \ NF \ B] \Rightarrow NF \ (\text{App} \ A \ B) \\
\text{fn: } & NF \ A \Rightarrow NF \ (Fn \ x \ T \ A) \\
\text{pair: } & [NF \ A; \ NF \ B] \Rightarrow NF \ (\text{Pair} \ A \ B) \\
\text{fst: } & [P \neq \text{Pair} \ A \ B; \ NF \ P] \Rightarrow NF \ (\text{Fst} \ P) \\
\text{snd: } & [P \neq \text{Pair} \ A \ B; \ NF \ P] \Rightarrow NF \ (\text{Snd} \ P)
\end{align*}

theorem progress:
assumes \( \Gamma \vdash M : \tau \)
shows \( NF \ M \lor (\exists M'. \ (M \rightarrow \beta \ M')) \)
using assms proof(induction \( M \) arbitrary: \( \Gamma \) \( \tau \) rule: trm-induct)
\begin{align*}
\text{case 1} & \quad \text{thus } ?\text{case using } NF,\text{unit by metis} \\
\text{next} \\
\text{case } (2 \ x) & \quad \text{thus } ?\text{case using } NF,\text{var by metis} \\
\text{next} \\
\text{case } (\beta \ A \ B) & \quad \text{from } \Gamma \vdash \text{App} \ A \ B : \tau \) obtain \( \sigma \\
& \quad \text{where } \Gamma \vdash A : (T\text{Arr} \, \sigma \, \tau) \text{ and } \Gamma \vdash B : \sigma \\
& \quad \text{using typing-appE by metis} \\
& \quad \text{hence } A: NF \ A \lor (\exists A'. \ (A \rightarrow \beta \ A')) \text{ and } B: NF \ B \lor (\exists B'. \ (B \rightarrow \beta \ B')) \\
& \quad \text{using 3.IH by auto} \\
\& \quad \text{consider } NF \ B \mid \exists B'. \ (B \rightarrow \beta \ B') \text{ using } B \text{ by auto}
\end{align*}
thus \(\text{case proof(cases)}\)

case 1
consider \(\text{NF} \ A \mid \exists A'. (A \rightarrow \beta A')\) using \(A\) by auto
thus \(\text{thesis proof(cases)}\)
  case 1
  consider \(\exists x \ T \ A'. A = \text{Fn} \ x \ T \ A' \mid \nexists x \ T \ A'. A = \text{Fn} \ x \ T \ A'\) by auto
  thus \(\text{thesis proof(cases)}\)
    case 1
    from this obtain \(x \ T \ A'\) where \(A = \text{Fn} \ x \ T \ A'\) by auto
    hence \((\text{App} \ A \ B) \rightarrow \beta (A'[x ::= B])\) using \text{beta-reduction.beta by metis}
    thus \(\text{thesis by blast}\)
  next
case 2
  thus \(\text{thesis using } \langle\text{NF} \ A; \langle\text{NF} \ B\rangle\rangle \text{NF.app by metis}\)
next
  qed
next
case 2
  thus \(\text{thesis using } \text{beta-reduction.app1 by metis}\)
next
  qed
next
case 2
  thus \(\text{thesis using } \text{beta-reduction.app2 by metis}\)
next
  qed
next
case 2
  thus \(\text{case proof(cases)}\)
  case 1
  thus \(\text{thesis using } \text{NF.fn by metis}\)
next
case 2
  from this obtain \(A'\) where \(A \rightarrow \beta A'\) by auto
  obtain \(M'\) where \(M' = \text{Fn} \ x \ T \ A'\) by auto
  hence \((\text{Fn} \ x \ T \ A) \rightarrow \beta M'\) using \(\langle A \rightarrow \beta A'\rangle\) \text{beta-reduction.fn by metis}
  thus \(\text{thesis by auto}\)
next
  qed
next
case 2
  thus \(\text{case using } \text{typing-pairE beta-reduction.pair1 beta-reduction.pair2 NF.pair}\)
by meson

next

case (6 P)
from this consider NF P \mid \exists P'. (P \rightarrow \beta P') using typing-fstE by metis
thus ?case proof(cases)
  case 1
    consider \exists A B. P = Pair A B | \nexists A B. P = Pair A B by auto
    thus ?thesis proof(cases)
      case 1
        from this obtain A B where P = Pair A B by auto
        hence (Fst P) \rightarrow \beta A using beta-reduction.fst2 by metis
        thus ?thesis by auto
      next
      case 2
      thus ?thesis using (NF P) NF.fst by metis
  next
next

case (7 P)
from this consider NF P \mid \exists P'. (P \rightarrow \beta P') using typing-sndE by metis
thus ?case proof(cases)
  case 1
    consider \exists A B. P = Pair A B | \nexists A B. P = Pair A B by auto
    thus ?thesis proof(cases)
      case 1
        from this obtain A B where P = Pair A B by auto
        hence (Snd P) \rightarrow \beta B using beta-reduction.snd2 by metis
        thus ?thesis by auto
      next
      case 2
      thus ?thesis using (NF P) NF.fst by metis
  next
next

inductive beta-reduces :: 'a trm \Rightarrow 'a trm \Rightarrow bool (\rightarrow \beta^* \cdot) where
  reflexive: M \rightarrow \beta^* M
  | transitive: [M \rightarrow \beta^* M'; M' \rightarrow \beta M''] \Rightarrow 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 if $fvs M' \subseteq fvs M$ 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 $(\text{Fn } x T A) \rightarrow \beta^* (\text{Fn } x T A')$
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 $(\text{Pair } A B) \rightarrow \beta^* (\text{Pair } A' B)$
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 $(\text{Pair } A B) \rightarrow \beta^* (\text{Pair } A' B')$
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 $(\text{Fst } P) \rightarrow \beta^* (\text{Fst } P')$
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 \( P \rightarrow \beta^* \textit{P}' \)
  shows \((\textit{Snd } P) \rightarrow \beta^* (\textit{Snd } P')\)
using assms proof (induction)
case (reflexive \( M \))
  thus ?case using beta-reduces.reflexive.
next
case (transitive \( M \textit{M}' \textit{M}'' \))
  thus ?case using beta-reduces.transitive beta-reduction.snd1 by metis
next
qed

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

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

inductive parallel-reduction :: \'a trm \Rightarrow \; \text{bool} \; (- >> -) \; where
  refl: \( A >> A \)
| beta: \[ A >> A'; B >> B' \] \( \Rightarrow \) 
  (\( \textit{App} \; (\textit{Fn} \; \; x \; \; T \; \; A) \; \; B \) \( \Rightarrow \) \( A'[x := B'] \))
| eta: \( A >> A' \) \( \Rightarrow \) 
  (\( \textit{Fn} \; \; x \; \; T \; \; A \) \( \Rightarrow \) \( A' \))
| app: \[ A >> A'; B >> B' \] \( \Rightarrow \) 
  (\( \textit{App} \; A \; B \) \( \Rightarrow \) \( A' \; B' \))
| pair: \[ A >> A'; B >> B' \] \( \Rightarrow \) 
  (\( \textit{Pair} \; A \; B \) \( \Rightarrow \) \( A' \; B' \))
| fst1: \( P >> P' \Rightarrow (\textit{Fst} \; P) \Rightarrow (\textit{Fst} \; P') \)
| fst2: \( A >> A' \Rightarrow (\textit{Fst} \; (\textit{Pair} \; A \; B)) \Rightarrow A' \)
| snd1: \( P >> P' \Rightarrow (\textit{Snd} \; P) \Rightarrow (\textit{Snd} \; P') \)
| snd2: \( B >> B' \Rightarrow (\textit{Snd} \; (\textit{Pair} \; A \; B)) \Rightarrow B' \)
lemma parallel-reduction-prm:
  assumes A >> A'
  shows \((\pi \cdot A) >> (\pi \cdot A')\)
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 >> 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 \(\ast\): \(fvs (App (Fn x T A) B) = fvs A - \{x\} \cup fvs B\) using fvs-simp(3, 4)
    by metis
    have \(fvs (A'[x := B']) \subseteq fvs A' - \{x\} \cup fvs B'\) using subst-fvs.
    also have ... \(\subseteq fvs A - \{x\} \cup fvs B\) 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 \textit{?case using} \texttt{fvs-simp(5, 7)} \textit{by force}

next
qed

\textbf{inductive-cases} \texttt{parallel-reduction-unitE'}: \texttt{Unit }>> \texttt{A}

\textbf{lemma} \texttt{parallel-reduction-unitE}:
- \texttt{assumes } \texttt{Unit }>> \texttt{A}
- \texttt{shows } \texttt{A }= \texttt{Unit}

\texttt{using} \texttt{assms}
- \texttt{apply (rule parallel-reduction-unitE'[where } \texttt{A=A])}
- \texttt{apply blast}
- \texttt{apply (auto simp add: unit-not-app unit-not-fn unit-not-pair unit-not-fst unit-not-snd)}
\texttt{done}

\textbf{inductive-cases} \texttt{parallel-reduction-varE'}: (\texttt{Var x}) >> \texttt{A}

\textbf{lemma} \texttt{parallel-reduction-varE}:
- \texttt{assumes } (\texttt{Var x}) >> \texttt{A}
- \texttt{shows } \texttt{A }= \texttt{Var x}

\texttt{using} \texttt{assms}
- \texttt{apply (rule parallel-reduction-varE'[where } \texttt{x=x and } \texttt{A=A])}
- \texttt{apply blast}
- \texttt{apply (auto simp add: var-not-app var-not-fn var-not-pair var-not-fst var-not-snd)}
\texttt{done}

\textbf{inductive-cases} \texttt{parallel-reduction-fnE'}: (\texttt{Fn x T A}) >> \texttt{X}

\textbf{lemma} \texttt{parallel-reduction-fnE}:
- \texttt{assumes } (\texttt{Fn x T A}) >> \texttt{X}
- \texttt{shows } \texttt{X }= \texttt{Fn x T A} \lor (\exists \texttt{A'T}. (\texttt{A }>> \texttt{A'}) \land \texttt{X }= \texttt{Fn x T A'})

\texttt{using} \texttt{assms proof(induction rule: parallel-reduction-fnE'[where } \texttt{x=x and } \texttt{T=T and } \texttt{A=A and } \texttt{X=X])}

\texttt{case } (\texttt{4 B B' y S})
- \texttt{from this consider } \texttt{x }= \texttt{y} \land \texttt{T }= \texttt{S} \land \texttt{A }= \texttt{B} \mid \texttt{x }\neq \texttt{y} \land \texttt{T }= \texttt{S} \land \texttt{x }\notin \texttt{fvs B} \land \texttt{A }= \texttt{\{x \leftrightarrow y\} \cdot B}

\texttt{using} \texttt{trm-simp(4) by metis}
- \texttt{thus } \texttt{?case proof(cases)}

\texttt{case 1}
- \texttt{hence } \texttt{x }= \texttt{y} \land \texttt{T }= \texttt{S} \land \texttt{A }= \texttt{B} \textit{ by auto}

\texttt{thus } \texttt{?thesis using } \texttt{4 by metis}

next

\texttt{case 2}
- \texttt{hence } \texttt{x }\neq \texttt{y} \land \texttt{T }= \texttt{S} \land \texttt{x }\notin \texttt{fvs B A }= \texttt{\{x \leftrightarrow y\} \cdot B} \textit{ by auto}

\texttt{hence } \texttt{x }\notin \texttt{fvs B' A }>> \texttt{([x \leftrightarrow y] \cdot B')} \textit{ by auto}

\texttt{using} \texttt{parallel-reduction-fvs parallel-reduction-prm \{B }>> \texttt{B'} \textit{ by auto}

\texttt{thus } \texttt{?thesis using} \texttt{fn-eq \{X }= \texttt{Fn y S B' y (x }\neq \texttt{y) \cdot T }= \texttt{S} \textit{ by metis}

next

\texttt{qed}

next
\texttt{qed (}
\texttt{metis assms,}

78
 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) >> X
lemma parallel-reduction-redexE:
  assumes (App (Fn x T A) B) >> X
  shows
  \((X = App (Fn x T A) B) \lor \\
  (\exists A' B'. (A >> A') \land (B >> B') \land X = (A'[x := B'])) \lor \\
  (\exists A' B'. ((Fn x T A) >> (Fn x T A')) \land (B >> B') \land X = (App (Fn x T A') B'))\)

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 (\App (Fn x T A) B = App C D) have C: C = Fn x T A and D: D = B
  by (metis trm-simp(2), metis trm-simp(3))
  from C and C' >> C' obtain A' where C': C' = Fn x T A'
  using parallel-reduction-fnE by metis
  thus ?thesis using C C' D (C >> C') (D >> D') \(X = App C' D') by metis
next
case (3 C C' D D' y S)
  from (\App (Fn x T A) B = App (Fn y S C) D) 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 \land T = S \land A = C
  | x \neq y \land T = S \land A = \([x \leftrightarrow y] \cdot C \land x \notin fvs C
  using trm-simp(4) by metis
  thus ?case proof (cases)
  case 1
  thus ?thesis using C >> C' \(X = (C'[y := D']) \langle D >> D' \rangle B by metis
next
case 2
  hence x \neq y \land T = S and A: A = \([x \leftrightarrow y] \cdot C x \notin fvs C by auto
  have x \notin fvs C' using parallel-reduction-fes \(x \notin fvs C' \langle C >> C' \rangle by force

  have A >> (\([x \leftrightarrow y] \cdot C')
    using parallel-reduction-prm \(C >> C' A by metis
  moreover have X = \((\([x \leftrightarrow y] \cdot C'[x := D'])
    using \(X = (C'[y := D']) : subst-sup \(x \notin fvs C' 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 \( \forall x T A' \). A \( \neq \) Fn x T A'
shows \( \exists A' B' \). (A >> A') \land (B >> B') \land 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') \land (B >> B') \land 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
\textbf{Qed} (  
\begin{itemize}
\item metis assms,  
\item metis app-not-pair,  
\item metis fn-not-pair,  
\item metis app-not-pair,  
\item metis pair-not-fst,  
\item metis pair-not-fst,  
\item metis pair-not-snd,  
\item metis pair-not-snd
\end{itemize}
)

\textbf{Inductive-Cases}  
\begin{itemize}
\item \texttt{parallel-reduction-fstE'}: (Fst P) \gg X
\end{itemize}

\textbf{Lemma}  
\begin{itemize}
\item \texttt{parallel-reduction-fstE}:  
\item \texttt{assumes} (Fst P) \gg X  
\item \texttt{shows} \((\exists P'. (P \gg P') \land X = (Fst P')) \lor (\exists A A' B. (P = Pair A B) \land (A \gg A') \land X = A'))  
\item \texttt{using} assms proof\((\text{induction rule: parallel-reduction-fstE'[where } P=P \text{ and } X=X])\)  
\item \texttt{case} (7 P P')  
\item \quad \texttt{thus} ?case using parallel-reduction.fst1 trm-simp(7) by metis  
\item \texttt{next}
\item \texttt{case} (8 A B)  
\item \quad \texttt{thus} ?case using parallel-reduction.fst2 trm-simp(7) by metis  
\item \texttt{next}
\end{itemize}
\textbf{Qed} (  
\begin{itemize}
\item metis assms,  
\item insert parallel-reduction.refl, metis,  
\item metis app-not-fst,  
\item metis fn-not-fst,  
\item metis app-not-fst,  
\item metis pair-not-fst,  
\item metis pair-not-snd,  
\item metis pair-not-snd
\end{itemize}
)

\textbf{Inductive-Cases}  
\begin{itemize}
\item \texttt{parallel-reduction-sndE'}: (Snd P) \gg X
\end{itemize}

\textbf{Lemma}  
\begin{itemize}
\item \texttt{parallel-reduction-sndE}:  
\item \texttt{assumes} (Snd P) \gg X  
\item \texttt{shows} \((\exists P'. (P \gg P') \land X = (Snd P')) \lor (\exists A B B'. (P = Pair A B) \land (B \gg B') \land X = B'))  
\item \texttt{using} assms proof\((\text{induction rule: parallel-reduction-sndE'[where } P=P \text{ and } X=X])\)  
\item \texttt{case} (9 P P')  
\item \quad \texttt{thus} ?case using parallel-reduction.snd1 trm-simp(8) by metis  
\item \texttt{next}
\item \texttt{case} (10 A B)  
\item \quad \texttt{thus} ?case using parallel-reduction.snd2 trm-simp(8) by metis  
\item \texttt{next}
\end{itemize}
\textbf{Qed} (  
\begin{itemize}
\item metis assms,  
\item insert parallel-reduction.refl, metis,
\end{itemize}
\begin{verbatim}
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 >> M'
  shows X[z := M] >> (X[z := M'])
using assms proof (induction X rule: trm-strong-induct[where S={z} \cup \text{fvs } M \cup \text{fvs } M'])
  show finite \((\{z\} \cup \text{fvs } M \cup \text{fvs } M')\) using fes-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 \text{fvs } M x \notin \text{fvs } M' by auto
    thus ?case using 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 >> X' M >> M'
  shows X[z := M] >> (X'[z := M'])
using assms proof (induction X arbitrary: X' rule: trm-strong-depth-induct[where S={z} \cup \text{fvs } M \cup \text{fvs } M'])
  show finite (\{z\} \cup \text{fvs } M \cup \text{fvs } M') using fes-finite by auto
  next
  case 1
\end{verbatim}
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 \( X' = \text{Var} \) using parallel-reduction-varE by metis
  thus \(?case using parallel-reduction-subst-inner \langle M >> M' \rangle by metis

next

case (3 C D)
  consider \( \exists x \ T A \cdot C = F n x \ T A \mid \exists x \ T A \cdot C = F n x \ T A \) by metis
  thus \(?case proof\(\langle cases\rangle
    case 1
    from this obtain \( x \ T A \) where \( C = F n x \ T A \) by auto
    have depth \( C < \) depth \( (\text{App} C D) \) depth \( D < \) depth \( (\text{App} C D) \)
      using depth-app by auto
    consider \( X' = \text{App} (F n x \ T A) D \)
    \[ \exists A' D'. ((F n x T A) >> (F n x T A')) \land (D >> D') \land X' = (\text{App} (F n x T A') D') \]
    using parallel-reduction-redexE \( \langle \text{App} C D \rangle >> X' \) \( C \) by metis
    thus \(?thesis proof\(\langle cases\rangle
      case 1
      thus \(?thesis using parallel-reduction-subst-inner \langle M >> M' \rangle \) \( C \) by metis
      next
      case 2
      from this obtain \( A' D' \)
      where \( (F n x T A) >> (F n x T A') D >> D' \) and \( X' = \text{App} (F n x T A') D' \)
      by auto
      have \(*\) : \((F n x T A)[z := M]\) >> \((F n x T A')[z := M']\)
      using 3.IH \langle depth C < depth (\text{App} C D) \rangle \langle (F n x T A) >> (F n x T A') D' \rangle \langle M >> M' \rangle by metis
      have \(*\) : \((D[z := M]) >> (D'[z := M'])\)
      using 3.IH \langle depth D < depth (\text{App} C D) \rangle \langle D >> D' \rangle \langle M >> M' \rangle by metis
      have \((\text{App} C D)[z := M] = \text{App} ((F n x T A)[z := M]) (D[z := M])\)
      using subst-simp-app C by metis
      moreover have \(... >> (\text{App} ((F n x T A')[z := M']) (D'[z := M']))\)
      using \(*\) parallel-reduction.app by metis
      moreover have \(... = ((\text{App} (F n x T A') D')[z := M'])\)
      using subst-simp-app by metis
      moreover have \(... = (X'[z := M']\)
      using \(X'\) by metis
      ultimately show \(?thesis by metis
      next
      case 3
from this obtain \( A' D' \) where \( A >> A' D >> D' \) and \( X' \): \( X' = \langle A'[x := D'] \rangle \)

by auto

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

have finite \( \{z\} \cup \text{fes} \ M \cup \text{fes} \ M' \cup \text{fes} \ A' \) using fes-finite by auto
from this obtain \( y \)
where \( y \notin \{z\} \cup \text{fes} \ M \cup \text{fes} \ M' \cup \text{fes} \ A' \) and \( C \): \( C = \text{Fn} \ y \ T \ (\langle y \leftrightarrow x \rangle \cdot A) \)

using fresh-fn \( C \) by metis
hence \( y \neq z \) \( y \notin \text{fes} \ M \ y \notin \text{fes} \ M' \ y \notin \text{fes} \ A' \) by auto
have \( (\langle y \leftrightarrow x \rangle \cdot A) >> (\langle y \leftrightarrow x \rangle \cdot A') \) using parallel-reduction-prm \( A \)

by metis

have \( (\text{App} \ C \ D)[z := M] = (\text{App} \ (\text{Fn} \ y \ T \ (\langle y \leftrightarrow x \rangle \cdot A))[z := M]) \)

(\( D[z := M] \))

using \( C \) subst-simp-app by metis
moreover have \( ... = (\text{App} \ (\text{Fn} \ y \ T \ (\langle y \leftrightarrow x \rangle \cdot A))[z := M]) \) \( (D[z := M]) \)

using \( \langle y \neq z \rangle \cdot y \notin \text{fes} \ M' \) subst-simp-fn by metis
moreover have \( ... >> (\langle y \leftrightarrow x \rangle \cdot A')[z := M'] \) \( (\langle y := D'[z := M'] \rangle \cdot y \leftrightarrow x \cdot A') \)

using parallel-reduction.beta \( \ast \ast \) by metis
moreover have \( ... = (\langle y \leftrightarrow x \rangle \cdot A')[y := D'[z := M']] \)

using barendregt \( \langle y \neq z \rangle \cdot y \notin \text{fes} \ M' \) by metis
moreover have \( ... = (A'[x := D][z := M']) \)

using subst-swap \( y \notin \text{fes} \ A' \) by metis
moreover have \( ... = (X'[z := M']) \) using \( X' \) by metis

ultimately show \( \? \)thesis by metis

next

qed

next
case 2

from this obtain \( C' D' \) where \( C >> C' D >> D' \) and \( X' \): \( X' = \text{App} \ C' D' \)

using parallel-reduction-nonredexE \( (\text{App} \ C \ D) >> X' \) by metis

have depth \( C < depth (\text{App} \ C \ D) \) depth \( D < depth (\text{App} \ C \ D) \)
using depth-app by auto
hence \( \ast \ast \): \( (C[z := M]) \text{ and } (D[z := M]) \text{ and } (\langle y \leftrightarrow x \rangle \cdot A') \)

using 3.IH \( : C >> C' \langle D >> D' \text{ by metis} + \)

84
have \((\text{App } C \ D)[z ::= M] = \text{App } (C[z ::= M]) (D[z ::= M])\)
using \text{subst-simp-app} by metis
moreover have \(\ldots >>(\text{App } (C'[z ::= M']) (D'[z ::= M']))\)
using \text{parallel-reduction.app} \(*\) by metis
moreover have \(\ldots = ((\text{App } C' \ D')[z ::= M'])\)
moreover have \(\ldots = (X'[z ::= M'])\) using \(X'\) by metis
ultimately show \(?\text{thesis}\) by metis
next
qed

next
case \((\_\_ x T A)\)
hence \(x \neq z\) \(\notin\) \text{fus } M \(\notin\) \text{fus } M'
by auto
from \((\text{Fn } x T A) >>(X'')\) consider
\(X'' = \text{Fn } x T A\)
| \(\exists A'. (A >> A') \land X' = \text{Fn } x T A'\) using \text{parallel-reduction-fnE} by metis
thus \(\text{?case}\) proof(cases)
case 1
  thus \(?\text{thesis}\) using \text{parallel-reduction-subst-inner} \(M >> M'\) by metis
next
case 2
  from this obtain \(A'\) where \(A >> A'\) and \(X' : X' = \text{Fn } x T A'\) by auto
  hence \(*\) : \((A[z ::= M]) >> (A'[z ::= M'])\)
  using \(4.\text{IH depth-fn } A >> A' : M >> M'\) by metis
have \(((\text{Fn } x T A)[z ::= M]) = (\text{Fn } x T (A[z ::= M]))\)
using \text{subst-simp-fn} \(x \neq z\) \(\notin\) \text{fus } M \(\notin\) \text{fus } M'
moreover have \(\ldots >>(\text{Fn } x T (A'[z ::= M']))\)
using \text{parallel-reduction.eta} \(*\) by metis
moreover have \(\ldots = ((\text{Fn } x T A')[z ::= M'])\)
moreover have \(\ldots = (X'[z ::= M'])\)
moreover have \(\ldots = (X'[z ::= M'])\)
ultimately show \(?\text{thesis}\) by metis
next
qed

next
case \((\_\_ x T A)\)
from \((\text{Pair } A B) >>(X'')\) consider
\(X'' = \text{Pair } A B\)
| \(\exists A'B'. (A >> A') \land (B >> B') \land X' = \text{Pair } A' B'\)
using \text{parallel-reduction-pairE} by metis
thus \(\text{?case}\) proof(cases)
case 1
  thus \(?\text{thesis}\) using \text{parallel-reduction-subst-inner} \(M >> M'\) by metis
next
case 2

from this obtain \( A' B' \) where \( A >> A' B >> B' \) and \( X' = Pair A' B' \) by auto

have \(*\): \((A[z := M]) >> (A'[z := M'])\) and \(**\): \((B[z := M]) >> (B'[z := M'])\)

using 5.IH \( \langle A >> A' \rangle \langle B >> B' \rangle \langle M >> M' \rangle \) 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 \(... = (\langle Pair A' B' \rangle[z := M'])\)

using subst-simp-pair by metis

moreover have \(... = (X'[z := M'])\) using \( X' \) by metis

ultimately show \(?thesis by metis\)

next

next
case (6 \( P \))

from \( \langle \text{Fst } P \rangle >> X' \) consider

\( \exists P'. (P >> P') \land X' = \text{Fst } P' \)

| \( \exists A A' B. \; P = \text{Pair } A B \land (A >> A') \land X' = A' \)

using parallel-reduction-fstE by metis

thus \(?case proof(cases)\)

case 1

from this obtain \( P' \) where \( P >> P' \) and \( X': X' = \text{Fst } P' \) by auto

have \(*\): \((P[z := M]) >> (P'[z := M'])\)

using 6.IH depth-fst \( \langle P >> P' \rangle \langle M >> M' \rangle \) by metis

have \((\text{Fst } P)[z := M] = \text{Fst } (P[z := M])\)

using subst-simp-fst by metis

moreover have \(... >> (\text{Fst } (P'[z := M']))\)

using parallel-reduction.fst1 \(*\ **\) by metis

moreover have \(... = (\langle \text{Fst } P' \rangle[z := M'])\)

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 \( P = \text{Pair } A B A >> A' \) and \( X': X' = A' \) by auto

have depth \( A < \text{depth } (\text{Fst } P) \)

using \( P \) depth-fst depth-pair dual-order.strict-trans by fastforce

hence \(*\): \((A[z := M]) >> (A'[z := M'])\)

86
using 6.IH \( \langle A \gg M \gg M \rangle \) by metis

have \((\text{Fst} \ P)[z := M] = (\text{Fst} \ (\text{Pair} \ (A[z := M]) \ (B[z := M])))\)
using \(P\) subst-simp-fst subst-simp-pair by metis
moreover have \(\ldots \gg (A'[z := M'])\)
using parallel-reduction.fst2 \(*\) by metis
moreover have \(\ldots = (X'[z := M'])\)
using \(X'\) by metis
ultimately show \(?thesis\) by metis
next
qed
next
case (7 \(P\))
from \((\text{Snd} \ P) \gg X'\) consider
\(\exists P', (P >> P') \land X' = \text{Snd} \ P'\)
| \(\exists A \ B \ B', P = \text{Pair} \ A \ B \land (B >> B') \land X' = B'\)
using parallel-reduction-sndE by metis
thus \(?case\) proof(cases)
case 1
from this obtain \(P'\) where \(P >> P'\) and \(X': X' = \text{Snd} \ P'\) by auto

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

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

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

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

inductive complete-development :: 'a trm ⇒ 'a trm ⇒ bool (- >>= -) where
  unit: Unit >>= Unit
  | var: (Var x) >>= (Var x)
  | beta: [A >>= A'; B >>= B'] ⇒ (App (Fn x T A) B) >>= (A'[x ::= B'])
  | eta: A >>= A' ⇒ (Fn x T A) >>= (Fn x T A')
  | app: [A >>= A'; B >>= B'; (x T M. A ≠ Fn x T M)] ⇒ (App A B) >>= (App A' B')
  | pair: [A >>= A'; B >>= B'] ⇒ (Pair A B) >>= (Pair A' B')
  | fst1: [P >>= P'; (A B. P ≠ Pair A B)] ⇒ (Fst P) >>= (Fst P')
  | fst2: A >>= A' ⇒ (Fst (Pair A B)) >>= A'
  | snd1: [P >>= P'; (A B. P ≠ Pair A B)] ⇒ (Snd P) >>= (Snd P')
  | snd2: B >>= B' ⇒ (Snd (Pair A B)) >>= B'

lemma not-fn-prm:
  assumes \( \forall x T. A \neq Fn x T M \)
  shows \( \pi \cdot A \neq Fn x T M \)
proof (rule classical)
  fix x T M
  obtain \( \pi' \) where \( \pi' = \text{prm-inv} \pi \) 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 \( (\pi' \cdot \pi) \cdot A = \pi' \cdot Fn x T M \)
    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 \( ?\text{thesis} \) by blast
qed

lemma not-pair-prm:
  assumes \( \forall A B. P \neq Pair A B \)
  shows \( \pi \cdot P \neq Pair A B \)
proof (rule classical)
  fix A B
  obtain \( \pi' \) where \( \pi' = \text{prm-inv} \pi \) 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 \( (\pi' \cdot \pi) \cdot P = \pi' \cdot (Pair A B) \)
    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 \texttt{trm-prm-simp}(3) by \texttt{metis}
hence False using \texttt{assms} by \texttt{blast}
thus \texttt{thesis} by \texttt{blast}

\textbf{qed}

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

\textbf{lemma} \texttt{complete-development-fvs}:
assumes $A >> A'$
shows $\text{fvs} \ A' \subseteq \text{fvs} \ A$

89
using assms proof (induction)
case unit
  thus ?case using fus-simp by auto
next
case (var x)
  thus ?case using fus-simp by auto
next
case (beta A A' B B' x T)
  have fus (A'[x := B']) ⊆ fus A' - {x} ∪ fus B' using subst-fus.
  also have ... ⊆ fus A - {x} ∪ fus B using beta.IH by auto
  also have ... ⊆ fus (Fn x T A) ∪ fus B using fus-simp(4) subset-refl by force
  also have ... ⊆ fus (App (Fn x T A) B) using fus-simp(3) subset-refl by force
  finally show ?case.
next
case (eta A A' x T)
  thus ?case using fus-simp(4) using Un-Diff subset-Un-eq by (metis (no-types, lifting))
next
case (app A A' B B')
  thus ?case using fus-simp(3) Un-mono by metis
next
case (pair A A' B B')
  thus ?case using fus-simp(5) Un-mono by metis
next
case (fst1 P P')
  thus ?case by (simp add: fus-simp(5), 6 sup.coboundedI1)
next
case (snd1 B B' A)
  thus ?case using fus-simp(5, 7) by fastforce
next
case (snd2 B B' A)
  thus ?case using fus-simp(5, 7) by fastforce
next qed

lemma complete-development-fnE:
assumes (Fn x T A) >>> X
shows ∃ A'. (A >>> A') ∧ 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 ∧ A = B | x ≠ y ∧ x ∉ fus B ∧ A = [x ↔ y] · 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 \(\text{?thesis}\) by auto

next

case 2
hence \(x \neq y\) \(x \notin \text{fvs} \, B\) and \(A: A = [x \leftrightarrow y] \cdot B\) by auto

hence \(x \notin \text{fvs} \, B'\) using \(B >>> B'\) \(\text{complete-development-fvs}\) by auto

obtain \(A'\) where \(A': A' = [x \leftrightarrow y] \cdot B'\) by auto

hence \(A >>> A'\) using \(A \, (B >>> B')\) \(\text{complete-development-prm}\) by auto

have \(X = Fn \, x \, T \, A'\)

using \(\text{fn-eq}\) \((x \neq y) \, (x \notin \text{fvs} \, B')\) \(A' \, (X = Fn \, y \, S \, B')\) \((T = S)\) by metis

thus \(\text{?thesis}\) using \((A >>> A')\) by auto

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,
  metis fn-not-snd
)

lemma complete-development-pairE:

assumes \((\text{Pair} \, A \, B) >>> X\)

shows \(\exists \, A' \, B'. \, (A >>> A') \land (B >>> B') \land X = \text{Pair} \, A' \, B'\)

using assms

apply cases

apply (metis unit-not-pair, metis var-not-pair, metis app-not-pair, metis fn-not-pair, metis fn-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 \(\exists \, X. \, (A >>> X)\)

proof\((\text{induction} \, A \, \text{rule: trm-induct})\)

  case 1
  
  obtain \(X : \, 'a \, \text{trm}\) where \(X = \text{Unit}\) by auto

  hence \(\text{Unit} >>> X\) using \(\text{complete-development} \, . \, \text{unit}\) by metis

  thus \(\text{?case}\) by auto

next

  case \((2 \, x)\)

  obtain \(X\) where \(X = \text{Var} \, x\) by auto
hence \((\text{Var} \ x) \ggg \ X\) using complete-development.var by metis
thus \(?case by auto\)
next
case \((3 \ A \ B)\)
  from this obtain \(A' \ B'\) where \(A':: A \ggg A'\) and \(B':: B \ggg B'\) by auto
  consider \((\exists x \ T \ M. \ A = \text{Fn} \ x \ T \ M) \mid \neg(\exists x \ T \ M. \ A = \text{Fn} \ x \ T \ M)\) by auto
  thus \(?case proof(cases)\)
    case 1
    from this obtain \(x \ T \ M\) where \(A:: A = \text{Fn} \ x \ T \ M\) by auto
    obtain \(X\) where \(X = (M'[:x := B'])\) by auto
    hence \((\text{App} \ A \ B) \ggg X\) by auto
    using complete-development.beta \((M \ggg M') \ B' A\) by metis
    thus \(?thesis by auto\)
  next
case 2
  thus \(?thesis using complete-development.app A' B' by metis\)
next
dqed
next
case \((4 \ x \ T \ A)\)
  from this obtain \(A'\) where \(A':: A \ggg A'\) by auto
  obtain \(X\) where \(X = \text{Fn} \ x \ T \ A'\) by auto
  hence \((\text{Fn} \ x \ T \ A) \ggg X\) 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 \(\exists A \ B. \ P = \text{Pair} \ A \ B \mid \exists A \ B. \ P = \text{Pair} \ A \ B\) by auto
  thus \(?case proof(cases)\)
    case 1
    from this obtain \(A \ B \ X\) where \(P = \text{Pair} \ A \ B \ P \ggg X\) using \(6\) by auto
      from this obtain \(A' \ B'\) where \(A \ggg A' \ B \ggg B' \ X = \text{Pair} \ A' \ B'\)
        using complete-development-pairE \(B'\) by metis
      thus \(?thesis using complete-development.fst2 \ P = \text{Pair} \ A \ B\) by metis
    next
case 2
      thus \(?thesis using complete-development.fst1 \ P = \text{Pair} \ A \ B\) by blast
    next
dqed
next
case \((7 \ P)\)
  consider \(\exists A \ B. \ P = \text{Pair} \ A \ B \mid \exists A \ B. \ P = \text{Pair} \ A \ B\) by auto
  thus \(?case proof(cases)\)
    case 1
from this obtain $A B X$ where $P = \text{Pair } A B P \triangleright\triangleright X$ using 7 by 

```isar
to obtain $A' B'$ where $A \triangleright\triangleright A' B \triangleright\triangleright B' X = \text{Pair } A' B'$
using complete-development-pairE by metis
thus ?thesis using complete-development.shtml (P = Pair A B) by metis
next
case 2
thus ?thesis using complete-development.shtml 7 by blast
next
qed
next

lemma complete-development-triangle:
assumes $A \triangleright\triangleright D A \triangleright\triangleright B$
shows $B \triangleright\triangleright D$
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 $A \triangleright\triangleright A' C \triangleright\triangleright C'$ using parallel-reduction.refl by metis+
from $(\text{App } (\text{Fn } x T A) C) \triangleright\triangleright B$ consider
  $B = \text{App } (\text{Fn } x T A) C$
  $\exists A'' C'' (A \triangleright\triangleright A'') \land (C \triangleright\triangleright C'') \land B = (A''[x := C''])$
  $\exists A'' C'' ((\text{Fn } x T A) \triangleright\triangleright (\text{Fn } x T A'')) \land (C \triangleright\triangleright C'') \land B = (\text{App } (\text{Fn } x T A'') C'')$
using parallel-reduction-redezE by metis
thus ?case proof (cases)
case 1
  thus ?thesis using parallel-reduction.beta $(A \triangleright\triangleright A' \langle C \triangleright\triangleright C' \rangle) \text{ by metis}$
next
case 2
  from this obtain $A'' C''$ where $A \triangleright\triangleright A'' C \triangleright\triangleright C''$ and B: B = (A''[x := C'']) by auto
    hence $A'' \triangleright\triangleright A' C'' \triangleright\triangleright C'$ using beta.IH by metis+
  thus ?thesis using B parallel-reduction-subst by metis
next
case 3
  from this obtain $A'' C''$
    where $(\text{Fn } x T A) \triangleright\triangleright (\text{Fn } x T A'') C \triangleright\triangleright C''$ and B: B = (App (Fn x T A'') C'')
  by auto
  hence $C'' \triangleright\triangleright C'$ using beta.IH by metis
  have $A'' \triangleright\triangleright A'$
  proof
next
qed
next
```
thm parallel-reduction-fnE
from ⟨(Fn x T A) >> (Fn x T A'')⟩ consider
Fn x T A = Fn x T A''
| ∃A'''. (A >> A''') ∧ Fn x T A''' = Fn x T A'''
using parallel-reduction-fnE by metis
hence A >> 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 >> 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
next
qed
thus ?thesis using beta.IH parallel-reduction.app by metis
next
case (eta A A' x T B)
  from this consider
  B = Fn x T A
  | ∃A'''. (A >> A''') ∧ 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 >> A'' and B = Fn x T A'' by auto
  thus ?thesis using eta.IH parallel-reduction.eta by metis
next
next
case (app A A' C C')
  from this obtain A'' C'' where A >> A'' C >> C'' and B = App A'' C''
using parallel-reduction-nonredexE by metis
hence A'' >> A' C'' >> C' using app.IH by metis+
thus ?case using B parallel-reduction.app by metis
next
case (pair A A' C C')
  from ⟨Pair A C⟩ >> B; and parallel-reduction-pairE obtain A'' C'' where
  A >> A'' C >> C'' B = Pair A'' C'' by metis
thus ?case using pair.IH parallel-reduction.pair by metis
next

94
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
  ∃ P''. (Pair A C) >> P'' ∧ B = Fst P''
  | ∃ A''. (A >> A'') ∧ (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 ⟨B = Fst P''⟩ 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
  ∃ P''. ((Pair A C) >> P'') ∧ B = Snd P''
  | ∃ C''. (C >> C'') ∧ (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 ⟨B = Snd P''⟩ 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

95
lemma \texttt{parallel-reduction-diamond}:
assumes $A >>> B \ A >>> C$
shows $\exists D. \ (B >>> D) \wedge (C >>> D)$
proof
obtain $D$ where $A >>>> D$ using complete-development-exists by metis
hence $(B >>> D) \wedge (C >>> D)$ using complete-development-triangle assms by auto
thus $\exists_D. ((B >>> D) \wedge (C >>> D))$ by blast
qed

inductive \texttt{parallel-reduces} :: \texttt{`a trm \Rightarrow `a trm \Rightarrow bool (- >>> * -)} where
reflexive: $A >>>* A$
transitive: $[ A >>>* A'; A' >>> A'' ] \Rightarrow A >>>* A''$

lemma \texttt{parallel-reduces-diamond}:
assumes $A >>>* B \ A >>>* C$
shows $\exists D. \ (B >>>* D) \wedge (C >>>* D)$
using assms proof (induction)
case (reflexive $A$)
thus $\exists_D. ((B >>>* D) \wedge (C >>>* D))$ by metis
next
case (transitive $A \ A' \ A''$)
from this obtain $C$ where $B >>>* C \ A' >>> C$ by metis
from $(A' >>> C) \ (A' >>> A'')$ obtain $D$ where $C >> D \ A'' >>> D$
using \texttt{parallel-reduction-diamond} by metis
thus $\exists_D. ((B >>>* D) \wedge (C >>>* D))$ by metis
next
qed

lemma \texttt{parallel-reduces-diamond}:
assumes $A >>>* B \ A >>>* C$
shows $\exists D. \ (B >>>* D) \wedge (C >>>* D)$
using assms proof (induction)
case (reflexive $A$)
thus $\exists_D. ((B >>>* D) \wedge (C >>>* D))$ by metis
next
case (transitive $A \ A' \ A''$)
from this obtain $C'$ where $A' >>>* C' \ C >>>* C'$ by metis
from this obtain $D$ where $A'' >>>* D \ C' >>> D$
using $(A' >>> A'') \ (A' >>>* C'')$ \texttt{parallel-reduces-diamond} by metis
thus $\exists_D. ((B >>>* D) \wedge (C >>>* C'))$ by metis
next
qed

lemma \texttt{beta-reduction-is-parallel-reduction}:
assumes $A \rightarrow_{\beta} B$
shows $A >>> B$
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 >> B
  shows A ->\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) ->\beta^* (App (Fn x T A') B')
      using (A ->\beta^* A')
    beta-reduces-fn beta-reduces-app-left beta-reduces-app-right beta-reduces-composition
    by metis
    moreover have (App (Fn x T A') B') ->\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 >>=* B) = (A →β* B)
proof –
  have →: (A >>=* B) ⇒ (A →β* B)
  proof(induction rule: parallel-reduces.induct)
    case (reflexive A)
      thus case using beta-reduces.reflective.
    next
    case (transitive A A' A'')
      thus case using beta-reduces-composition parallel-reduction-is-beta-reduction by metis
  next
  qed

have ←: (A →β* B) ⇒ (A >>=* B)
proof(induction rule: beta-reduces.induct)
  case (reflexive A)
    thus case using parallel-reduces.reflective.
  next
  case (transitive A A' A'')
    thus case using parallel-reduces.composition transitive beta-reduction-is-parallel-reduction by metis
  next
  qed

from ←→ show (A >>=* B) = (A →β* B) by blast
qed

theorem confluence:
  assumes A →β* B A →β* C
  shows ∃D. (B →β* D) ∧ (C →β* D)
proof –
  from assms have A >>=* B A >>=* C using parallel-beta-reduces-equivalent by metis+
  hence ∃D. (B >>=* D) ∧ (C >>=* D) using parallel-reduces-diamond by metis
  thus ∃D. (B →β* D) ∧ (C →β* D) using parallel-beta-reduces-equivalent by metis
  qed

end
end