Verified Metatheory and Type Inference for a Name-Carrying Simply-Typed λ-Calculus

Michael Rawson

September 13, 2023

Abstract

I formalise a Church-style simply-typed λ-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* $\Longrightarrow$ *fresh-in S* $\notin$ *S*

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

  instance proof
    fix *S* :: *nat set*
    assume *finite S*
    consider *Set.is-empty S* | $\neg$*Set.is-empty S* **by** *auto*
    thus *fresh-in S* $\notin$ *S* **unfolding** *fresh-in-nat-def*
    proof(*cases*)
      case *1*
        hence *S* = {} **using** *Set.is-empty-def* **by** *metis*
        hence *0* $\notin$ *S* **by** *auto*
        thus (*if Set.is-empty S then 0 else Max S* + *1*) $\notin$ *S* **using** *1* **by** *auto*
      next

```
      case 2
        have Max S + 1 ∉ 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) ∉ S using 2 by auto
      next
    qed
  qed
end

end
theory Permutation
imports Main
begin

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

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

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

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

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

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

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

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

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

lemma preprm-ext-reflexive:
  shows x =p x
unfolding preprm-ext-def by auto

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

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

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

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

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

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

lemma *preprm-apply-unequal*:
  assumes $x \neq y$
  shows *preprm-apply* $\pi\ x \neq$ *preprm-apply* $\pi\ y$
using *assms* **proof**(*induction* π, *simp*)
  case (*Cons s ss*)
    have *preprm-apply* (*s* # *ss*) x = *swp-apply s* (*preprm-apply ss x*)
      and *preprm-apply* (*s* # *ss*) y = *swp-apply s* (*preprm-apply ss y*) **by** *auto*
    thus *?case* **using** *Cons.IH* ‹$x \neq y$› *swp-apply-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 *s1 s2* **where** *s1* = *fst s s2* = *snd s* **by** *auto*
hence *s* = (*s1*, *s2*) **by** *simp*
thus ∀ *x*. *preprm-apply* ([*s*] @ [*s*]) *x* = *preprm-apply* [] *x*
by *simp*
qed

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

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

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

lemma *preprm-disagreement-composition*:
assumes *a* ≠ *b b* ≠ *c a* ≠ *c*
shows {*x*.
preprm-apply (*preprm-compose* (*preprm-unit a b*) (*preprm-unit b c*)) *x* ≠
preprm-apply (*preprm-unit a c*) *x*
} = {*a*, *b*}
unfolding *preprm-unit-def preprm-compose-def* **proof**
show {*x*. *preprm-apply* ([(*a*, *b*)] @ [(*b*, *c*)]) *x* ≠ *preprm-apply* [(*a*, *c*)] *x*} ⊆ {*a*, *b*}
proof
fix *x*
have *x* ∉ {*a*, *b*} ⟹ *x* ∉ {*x*. *preprm-apply* ([(*a*, *b*)] @ [(*b*, *c*)]) *x* ≠ *preprm-apply* [(*a*, *c*)] *x*}
proof −
assume *x* ∉ {*a*, *b*}
hence *x* ≠ *a* ∧ *x* ≠ *b* **by** *auto*

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

lemma *preprm-compose-push*:
shows
preprm-compose π (*preprm-unit* a b) =*p*
preprm-compose (*preprm-unit* (*preprm-apply* π a) (*preprm-apply* π b)) π

unfolding *preprm-ext-def preprm-unit-def*
by (*simp add*: *inj-eq preprm-apply-composition preprm-apply-injective*)

lemma *preprm-ext-compose-left*:
assumes P =*p* S
shows *preprm-compose* π P =*p* *preprm-compose* π S
using *assms* **unfolding** *preprm-ext-def*
using *preprm-apply-composition* **by** *metis*

lemma *preprm-ext-compose-right*:
assumes P =*p* S
shows *preprm-compose* P π =*p* *preprm-compose* S π
using *assms* **unfolding** *preprm-ext-def*
using *preprm-apply-composition* **by** *metis*

lemma *preprm-ext-uncompose*:
assumes π =*p* σ *preprm-compose* π P =*p* *preprm-compose* σ S
shows P =*p* S
using *assms* **unfolding** *preprm-ext-def*
proof −
assume $*$: $\forall x.$ *preprm-apply* π x = *preprm-apply* σ x

assume $\forall x.$ *preprm-apply* (*preprm-compose* π P) x = *preprm-apply* (*preprm-compose* σ S) x
hence $\forall x.$ *preprm-apply* π (*preprm-apply* P x) = *preprm-apply* σ (*preprm-apply*

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

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

  have (*preprm-compose* (*rev* (*p* # *ps*)) (*p* # *ps*)) =*p* (*preprm-compose* (*rev ps*) (*preprm-compose* (*preprm-compose* [*p*] [*p*]) *ps*))
   unfolding *preprm-compose-def* **using** *preprm-ext-reflexive* **by** *simp*
  moreover have ... =*p* (*preprm-compose* (*rev ps*) (*preprm-compose preprm-id ps*))
  using *preprm-singleton-involution preprm-ext-compose-left preprm-ext-compose-right* **by** *metis*
  moreover have ... =*p* (*preprm-compose* (*rev ps*) *ps*)
   unfolding *preprm-compose-def preprm-id-def* **using** *preprm-ext-reflexive* **by** *simp*
  moreover have ... =*p preprm-id* **using** *IH***.**
  ultimately show *?case* **using** *preprm-ext-transitive* **by** *metis*
 next
qed

lemma *preprm-inv-involution*:
 shows *preprm-inv* (*preprm-inv* π) = π
unfolding *preprm-inv-def* **by** *simp*

lemma *preprm-inv-ext*:
 assumes π =*p* σ
 shows *preprm-inv* π =*p preprm-inv* σ
proof −
 have
  (*preprm-compose* (*preprm-inv* (*preprm-inv* π)) (*preprm-inv* π)) =*p preprm-id*
  (*preprm-compose* (*preprm-inv* (*preprm-inv* σ)) (*preprm-inv* σ)) =*p preprm-id*
  using *preprm-inv-compose* **by** *metis*+
 hence
  (*preprm-compose* π (*preprm-inv* π)) =*p preprm-id*
  (*preprm-compose* σ (*preprm-inv* σ)) =*p preprm-id*
  using *preprm-inv-involution* **by** *metis*+
 hence (*preprm-compose* π (*preprm-inv* π)) =*p* (*preprm-compose* σ (*preprm-inv* σ))

  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 equivpI*)
 show *reflp preprm-ext* **using** *preprm-ext-reflp*.
 show *symp preprm-ext* **using** *preprm-ext-symp*.
 show *transp preprm-ext* **using** *preprm-ext-transp*.
qed

lift-definition *prm-id* :: *'a prm* (ε) **is** *preprm-id*.

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

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

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

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

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

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

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

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

by (*transfer*, *metis preprm-unit-inaction*)

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

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

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

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

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

lemma *prm-inv-compose*:
  shows (*prm-inv* π) $\diamond\ \pi = \varepsilon$
by(*transfer*, *metis preprm-inv-compose*)

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

interpretation *'a prm*: *group prm-compose prm-id prm-inv*
unfolding *group-def group-axioms-def*
proof −
  have *semigroup* ($\diamond$) **using** *'a prm.semigroup-axioms*.
  moreover have $\forall a.\ \varepsilon \diamond a = a$ **by**(*transfer*, *simp add*: *preprm-id-def preprm-compose-def preprm-ext-def*)
  moreover have $\forall a.$ *prm-inv* $a \diamond a = \varepsilon$ **using** *prm-inv-compose* **by** *blast*
  ultimately show *semigroup* ($\diamond$) $\wedge$ ($\forall a.\ \varepsilon \diamond a = a$) $\wedge$ ($\forall a.$ *prm-inv* $a \diamond a = \varepsilon$)
by *blast*
qed

definition *prm-set* :: *'a prm* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* (**infix** {\$} *140*) **where**
  prm-set π $S \equiv$ *image* (*prm-apply* π) S

lemma *prm-set-apply-compose*:
  shows π {\$} (σ {\$} S) = ($\pi \diamond \sigma$) {\$} S

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

end
theory *PreSimplyTyped*
imports *Fresh Permutation*
begin

type-synonym *tvar* = *nat*

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

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

| $PFst$ $'a$ $ptrm$
| $PSnd$ $'a$ $ptrm$

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

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

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

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

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

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

lemma *ptrm-size-prm*:
  shows *size* X = *size* $(\pi \cdot X)$
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 ... = π {\$} $\{x\}$ **using** *prm-set-singleton* **by** *metis*
    moreover have ... = π {\$} *ptrm-fvs* (*PVar* x) **by** *simp*
    ultimately show *?case* **by** *metis*
  next
  case (*PApp A B*)
    have *ptrm-fvs* $(\pi \cdot$ *PApp* A $B)$ = *ptrm-fvs* (*PApp* $(\pi \cdot A)$ $(\pi \cdot B)$) **by** *simp*
    moreover have ... = *ptrm-fvs* $(\pi \cdot A) \cup$ *ptrm-fvs* $(\pi \cdot B)$ **by** *simp*
    moreover have ... = π {\$} *ptrm-fvs* $A \cup \pi$ {\$} *ptrm-fvs* B **using** *PApp.IH* **by**
metis
    moreover have ... = π {\$} (*ptrm-fvs* $A \cup$ *ptrm-fvs* B) **using** *prm-set-distributes-union*
by *metis*
    moreover have ... = π {\$} *ptrm-fvs* (*PApp* A B) **by** *simp*
    ultimately show *?case* **by** *metis*
  next
  case (*PFn x T A*)
    have *ptrm-fvs* $(\pi \cdot$ *PFn* x T $A)$ = *ptrm-fvs* (*PFn* $(\pi$ \$ $x)$ T $(\pi \cdot A)$) **by** *simp*
    moreover have ... = *ptrm-fvs* $(\pi \cdot A) - \{\pi$ \$ $x\}$ **by** *simp*
    moreover have ... = π {\$} *ptrm-fvs* $A - \{\pi$ \$ $x\}$ **using** *PFn.IH* **by** *metis*
    moreover have ... = π {\$} *ptrm-fvs* $A - \pi$ {\$} $\{x\}$ **using** *prm-set-singleton*
by *metis*
    moreover have ... = π {\$} (*ptrm-fvs* $A - \{x\}$) **using** *prm-set-distributes-difference*

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

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

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

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

lemma *ptrm-alpha-equiv-fvs-transfer*:
 assumes $A \approx [a \leftrightarrow b] \cdot B$ **and** $a \notin$ *ptrm-fvs B*
 shows $b \notin$ *ptrm-fvs A*
proof −
 from ‹$A \approx [a \leftrightarrow b] \cdot B$› **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** ‹$a \notin$ *ptrm-fvs B*› **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 $\bigwedge a.\ a \in ds\ \pi\ \sigma \Longrightarrow a \notin$ *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 $x \in ds\ \pi\ \sigma \mid x \notin ds\ \pi\ \sigma$ **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

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

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

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

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

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

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

case (*fn2 a b A B T*)
have $b \neq a$ **using** ‹$a \neq b$› **by** *auto*
have $B \approx [b \leftrightarrow a] \cdot A$ **using** ‹$[a \leftrightarrow b] \cdot B \approx A$›
using *ptrm-swp-transfer prm-unit-commutes* **by** *metis*

have $b \notin$ *ptrm-fvs A* **using** ‹$a \notin$ *ptrm-fvs B*› ‹$A \approx [a \leftrightarrow b] \cdot B$› ‹$a \neq b$›
using *ptrm-alpha-equiv-fvs-transfer* **by** *metis*

show *?case* **using** ‹$b \neq a$› ‹$B \approx [b \leftrightarrow a] \cdot A$› ‹$b \notin$ *ptrm-fvs A*›
using *ptrm-alpha-equiv.fn2* **by** *metis*
next
qed

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

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

have *size A < size X* **and** *size B < size X* **using** ‹$X =$ *PApp A B*› **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** ‹$X =$ *PApp A B*› ‹$Z =$ *PApp E F*› *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 $Y\text{-}cases$: $(x = y \wedge A \approx B) \vee (x \neq y \wedge A \approx [x \leftrightarrow y] \cdot B \wedge x \notin ptrm\text{-}fvs\ B)$
using fnE ‹$X \approx Y$› ‹$X = PFn\ x\ T\ A$› **by** *metis*
obtain $z\ C$ **where** Z: $Z = PFn\ z\ T\ C$
and $Z\text{-}cases$: $(y = z \wedge B \approx C) \vee (y \neq z \wedge B \approx [y \leftrightarrow z] \cdot C \wedge y \notin ptrm\text{-}fvs\ C)$
using fnE ‹$Y \approx Z$› ‹$Y = PFn\ y\ T\ B$› **by** *metis*

consider
$x = y\ A \approx B$ **and** $y = z\ B \approx C$
| $x = y\ A \approx B$ **and** $y \neq z\ B \approx [y \leftrightarrow z] \cdot C\ y \notin ptrm\text{-}fvs\ C$
| $x \neq y\ A \approx [x \leftrightarrow y] \cdot B\ x \notin ptrm\text{-}fvs\ B$ **and** $y = z\ B \approx C$
| $x \neq y\ A \approx [x \leftrightarrow y] \cdot B\ x \notin ptrm\text{-}fvs\ B$ **and** $y \neq z\ B \approx [y \leftrightarrow z] \cdot C\ y \notin ptrm\text{-}fvs\ C$ **and** $x = z$
| $x \neq y\ A \approx [x \leftrightarrow y] \cdot B\ x \notin ptrm\text{-}fvs\ B$ **and** $y \neq z\ B \approx [y \leftrightarrow z] \cdot C\ y \notin ptrm\text{-}fvs\ C$ **and** $x \neq z$
using $Y\text{-}cases$ $Z\text{-}cases$ **by** *auto*

thus *?thesis* **proof**(*cases*)
case *1*
have $A \approx C$ **using** ‹$A \approx B$› ‹$B \approx C$› IH $*$ **by** *metis*
have $x = z$ **using** ‹$x = y$› ‹$y = z$› **by** *metis*
show *?thesis* **using** ‹$A \approx C$› ‹$x = z$› X Z
using *ptrm-alpha-equiv.fn1* **by** *metis*
next
case *2*
have $x \neq z$ **using** ‹$x = y$› ‹$y \neq z$› **by** *metis*
have $A \approx [x \leftrightarrow z] \cdot C$ **using** ‹$A \approx B$› ‹$B \approx [y \leftrightarrow z] \cdot C$› ‹$x = y$› IH $*$ **by** *metis*
have $x \notin ptrm\text{-}fvs\ C$ **using** ‹$x = y$› ‹$y \notin ptrm\text{-}fvs\ C$› **by** *metis*
thus *?thesis* **using** ‹$x \neq z$› ‹$A \approx [x \leftrightarrow z] \cdot C$› ‹$x \notin ptrm\text{-}fvs\ C$› X Z
using *ptrm-alpha-equiv.fn2* **by** *metis*
next
case *3*
have $x \neq z$ **using** ‹$x \neq y$› ‹$y = z$› **by** *metis*
have $[x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot C$ **using** ‹$B \approx C$› *ptrm-alpha-equiv-prm* **by** *metis*
have $A \approx [x \leftrightarrow z] \cdot C$
using ‹$A \approx [x \leftrightarrow y] \cdot B$› ‹$[x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot C$› ‹$y = z$› IH $*$
by *metis*
have $x \notin ptrm\text{-}fvs\ C$ **using** ‹$B \approx C$› ‹$x \notin ptrm\text{-}fvs\ B$› *ptrm-alpha-equiv-fvs* **by** *metis*

thus *?thesis* **using** ‹$x \neq z$› ‹$A \approx [x \leftrightarrow z] \cdot C$› ‹$x \notin$ *ptrm-fvs* C› X Z
using *ptrm-alpha-equiv.fn2* **by** *metis*
next
case *4*
have $[x \leftrightarrow y] \cdot B \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C$
using ‹$B \approx [y \leftrightarrow z] \cdot C$› *ptrm-alpha-equiv-prm* **by** *metis*
hence $A \approx [x \leftrightarrow y] \cdot [y \leftrightarrow z] \cdot C$
using ‹$A \approx [x \leftrightarrow y] \cdot B$› *IH* $*$ **by** *metis*
hence $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C$ **using** *ptrm-prm-apply-compose* **by** *metis*
hence $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow x]) \cdot C$ **using** ‹$x = z$› **by** *metis*
hence $A \approx ([x \leftrightarrow y] \diamond [x \leftrightarrow y]) \cdot C$ **using** *prm-unit-commutes* **by** *metis*
hence $A \approx \varepsilon \cdot C$ **using** ‹$x = z$› *prm-unit-involution* **by** *metis*
hence $A \approx C$ **using** *ptrm-prm-apply-id* **by** *metis*

thus *?thesis* **using** ‹$x = z$› ‹$A \approx C$› X Z
using *ptrm-alpha-equiv.fn1* **by** *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** *metis*
hence $x \notin$ *ptrm-fvs* $([y \leftrightarrow z] \cdot C)$ **using** ‹$x \notin$ *ptrm-fvs* B› **by** *auto*
hence $x \notin [y \leftrightarrow z]$ {\$} *ptrm-fvs* C **using** *ptrm-prm-fvs* **by** *metis*
consider $z \in$ *ptrm-fvs* C | $z \notin$ *ptrm-fvs* C **by** *auto*
thus *?thesis* **proof**(*cases*)
case *1*
hence $[y \leftrightarrow z]$ {\$} *ptrm-fvs* $C =$ *ptrm-fvs* $C - \{z\} \cup \{y\}$
using *prm-set-unit-action prm-unit-commutes*
and ‹$z \in$ *ptrm-fvs* C› ‹$y \notin$ *ptrm-fvs* C› **by** *metis*
hence $x \notin$ *ptrm-fvs* $C - \{z\} \cup \{y\}$ **using** ‹$x \notin [y \leftrightarrow z]$ {\$} *ptrm-fvs* C› **by** *auto*
hence $x \notin$ *ptrm-fvs* $C - \{z\}$ **using** ‹$x \neq y$› **by** *auto*
thus *?thesis* **using** ‹$x \neq z$› **by** *auto*
next
case *2*
hence $[y \leftrightarrow z]$ {\$} *ptrm-fvs* $C =$ *ptrm-fvs* C **using** *prm-set-unit-inaction* ‹$y \notin$ *ptrm-fvs* C› **by** *metis*
thus *?thesis* **using** ‹$x \notin [y \leftrightarrow z]$ {\$} *ptrm-fvs* C› **by** *auto*
next
qed
qed

have $A \approx [x \leftrightarrow z] \cdot C$ **proof** $-$
have $A \approx ([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C$
using *IH* $*$ ‹$A \approx [x \leftrightarrow y] \cdot B$› ‹$B \approx [y \leftrightarrow z] \cdot C$›
and *ptrm-alpha-equiv-prm ptrm-prm-apply-compose* **by** *metis*

have $([x \leftrightarrow y] \diamond [y \leftrightarrow z]) \cdot C \approx [x \leftrightarrow z] \cdot C$ **proof** $-$

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

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

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

show *?thesis* **using** $\langle x \neq z\rangle$ $\langle A \approx [x \leftrightarrow z] \cdot C\rangle$ $\langle x \notin$ *ptrm-fvs* $C\rangle$ X Z
using *ptrm-alpha-equiv.fn2* **by** *metis*
next
qed
next
case (*PPair* A B)
obtain C D **where** $Y =$ *PPair* C D **and** $A \approx C$ **and** $B \approx D$
using *pairE* $\langle X =$ *PPair* A $B\rangle$ $\langle X \approx Y\rangle$ **by** *metis*
hence *PPair* C $D \approx Z$ **using** $\langle Y \approx Z\rangle$ **by** *simp*
from *this* **obtain** E F **where** $Z =$ *PPair* E F **and** $C \approx E$ **and** $D \approx F$ **using** *pairE* **by** *metis*

have *size* $A <$ *size* X **and** *size* $B <$ *size* X **using** $\langle X =$ *PPair* A $B\rangle$ **by** *auto*
hence $A \approx E$ **and** $B \approx F$ **using** *IH* $\langle A \approx C\rangle$ $\langle C \approx E\rangle$ $\langle B \approx D\rangle$ $\langle D \approx F\rangle$ **by** *auto*
thus *?thesis* **using** $\langle X =$ *PPair* A $B\rangle$ $\langle Z =$ *PPair* E $F\rangle$ *ptrm-alpha-equiv.pair* **by** *metis*
next
case (*PFst* P)
obtain Q **where** $Y =$ *PFst* Q $P \approx Q$ **using** *fstE* $\langle X =$ *PFst* $P\rangle$ $\langle X \approx Y\rangle$ **by** *metis*
obtain R **where** $Z =$ *PFst* R $Q \approx R$ **using** *fstE* $\langle Y =$ *PFst* $Q\rangle$ $\langle Y \approx Z\rangle$ **by** *metis*

have *size* $P <$ *size* X **using** $\langle X =$ *PFst* $P\rangle$ **by** *auto*
hence $P \approx R$ **using** *IH* $\langle P \approx Q\rangle$ $\langle Q \approx R\rangle$ **by** *metis*
thus *?thesis* **using** $\langle X =$ *PFst* $P\rangle$ $\langle Z =$ *PFst* $R\rangle$ *ptrm-alpha-equiv.fst* **by** *metis*
next
case (*PSnd* P)
obtain Q **where** $Y =$ *PSnd* Q $P \approx Q$ **using** *sndE* $\langle X =$ *PSnd* $P\rangle$ $\langle X \approx Y\rangle$ **by** *metis*
obtain R **where** $Z =$ *PSnd* R $Q \approx R$ **using** *sndE* $\langle Y =$ *PSnd* $Q\rangle$ $\langle Y \approx Z\rangle$

by *metis*

have *size* $P <$ *size* X **using** ‹$X = PSnd\ P$› **by** *auto*
hence $P \approx R$ **using** *IH* ‹$P \approx Q$› ‹$Q \approx R$› **by** *metis*
thus *?thesis* **using** ‹$X = PSnd\ P$› ‹$Z = PSnd\ R$› *ptrm-alpha-equiv.snd* **by** *metis*
next
qed
qed

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

type-synonym *'a typing-ctx* = *'a* $\rightharpoonup$ *type*

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

```
    thus ?case by simp
  next
  case (PVar x)
    consider x = a | x = b | x ≠ a ∧ x ≠ b by auto
    thus ?case proof(cases)
      assume x = a
      thus ?thesis using ‹a ≠ b› by (simp add: 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 ↦ T, a ↦ τ)) A
of None ⇒ None | Some σ ⇒ Some (TArr τ σ))
          using ‹a ≠ b› fun-upd-twist fun-upd-upd by metis
        moreover have ... = ptrm-infer-type (Γ(b ↦ S, a ↦ T)) (PFn x τ A)
          using ‹x = a› by simp
        moreover have ... = ptrm-infer-type (Γ(a ↦ T, b ↦ S)) (PFn x τ A)
```

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

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

lemma *ptrm-infer-type-swp*:
assumes $a \neq b$ $b \notin$ *ptrm-fvs* X
shows *ptrm-infer-type* $(\Gamma(a \mapsto \tau))$ $X =$ *ptrm-infer-type* $(\Gamma(b \mapsto \tau))$ $([a \leftrightarrow b] \cdot X)$
using *assms* **proof**(*induction X arbitrary*: τ Γ)
case $(PUnit)$
thus *?case* **by** *simp*
next
case $(PVar\ x)$
hence $x \neq b$ **by** *simp*
consider $x = a \mid x \neq a$ **by** *auto*
thus *?case* **proof**(*cases*)
case *1*
hence $[a \leftrightarrow b] \cdot (PVar\ x) = PVar\ b$
and *ptrm-infer-type* $(\Gamma(a \mapsto \tau))$ $(PVar\ x) = Some\ \tau$ **using** *prm-unit-action* **by** *auto*
thus *?thesis* **by** *auto*
next

case *2*
hence $*$: $[a \leftrightarrow b] \cdot PVar\ x = PVar\ x$ **using** $\langle x \neq b \rangle$ *prm-unit-inaction* **by** *simp*
consider $\exists\sigma.\ \Gamma\ x = Some\ \sigma \mid \Gamma\ x = None$ **by** *auto*
thus *?thesis* **proof**(*cases*)
assume $\exists\sigma.\ \Gamma\ x = Some\ \sigma$
from *this* **obtain** σ **where** $\Gamma\ x = Some\ \sigma$ **by** *auto*
thus *?thesis* **using** $*$ $\langle x \neq a \rangle$ $\langle x \neq b \rangle$ **by** *auto*
next

assume $\Gamma\ x = None$
thus *?thesis* **using** $*$ $\langle x \neq a \rangle$ $\langle x \neq b \rangle$ **by** *auto*
qed
next
qed
next
case $(PApp\ A\ B)$

have $b \notin$ *ptrm-fvs* A **and** $b \notin$ *ptrm-fvs* B **using** *PApp.prems* **by** *auto*
hence *ptrm-infer-type* $(\Gamma(a \mapsto \tau))$ $A =$ *ptrm-infer-type* $(\Gamma(b \mapsto \tau))$ $([a \leftrightarrow b] \cdot A)$
and *ptrm-infer-type* $(\Gamma(a \mapsto \tau))$ $B =$ *ptrm-infer-type* $(\Gamma(b \mapsto \tau))$ $([a \leftrightarrow b] \cdot B)$
using *PApp.IH assms* **by** *metis+*

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

case *2*
hence
ptrm-infer-type $(\Gamma(b \mapsto \tau))$ $([a \leftrightarrow b] \cdot$ *PFn* x σ $A)$
$=$ *ptrm-infer-type* $(\Gamma(b \mapsto \tau))$ (*PFn* x σ $([a \leftrightarrow b] \cdot A)$)
using ‹$b \neq x$› **by** (*simp add*: *prm-unit-inaction*)
moreover have ... $=$ (*case ptrm-infer-type* $(\Gamma(b \mapsto \tau,\ x \mapsto \sigma))$ $([a \leftrightarrow b] \cdot A)$ *of None* $\Rightarrow$ *None* | *Some* σ' $\Rightarrow$ *Some* (*TArr* σ σ'))
by *simp*
moreover have ... $=$ (*case ptrm-infer-type* $(\Gamma(x \mapsto \sigma,\ b \mapsto \tau))$ $([a \leftrightarrow b] \cdot A)$ *of None* $\Rightarrow$ *None* | *Some* σ' $\Rightarrow$ *Some* (*TArr* σ σ'))
using ‹$b \neq x$› *fun-upd-twist* **by** *metis*
moreover have ... $=$ (*case ptrm-infer-type* $(\Gamma(x \mapsto \sigma,\ a \mapsto \tau))$ A *of None* $\Rightarrow$ *None* | *Some* σ' $\Rightarrow$ *Some* (*TArr* σ σ'))
using $*$ **by** *metis*
moreover have ... $=$ (*case ptrm-infer-type* $(\Gamma(a \mapsto \tau,\ x \mapsto \sigma))$ A *of None* $\Rightarrow$ *None* | *Some* σ' $\Rightarrow$ *Some* (*TArr* σ σ'))
using ‹$a \neq x$› *fun-upd-twist* **by** *metis*
moreover have ... $=$ *ptrm-infer-type* $(\Gamma(a \mapsto \tau))$ (*PFn* x σ A)
by *simp*
ultimately show *?thesis* **by** *metis*

```
        next
      qed
    next

    case 2
      hence a ≠ x using assms by auto
      have
        ptrm-infer-type (Γ(a ↦ τ, b ↦ σ)) A =
        ptrm-infer-type (Γ(b ↦ τ, a ↦ σ)) ([a ↔ b] · A)
       using ptrm-infer-type-swp-types using ‹a ≠ 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 (PPair A B)
  thus ?case by simp
 next
 case (PFst P)
  thus ?case by simp
 next
 case (PSnd P)
  thus ?case by simp
 next
qed

lemma ptrm-infer-type-alpha-equiv:
 assumes X ≈ Y
 shows ptrm-infer-type Γ X = ptrm-infer-type Γ Y
using assms proof(induction arbitrary: Γ)
 case (fn2 a b A B T Γ)
  hence ptrm-infer-type (Γ(a ↦ T)) A = ptrm-infer-type (Γ(b ↦ T)) B
   using ptrm-infer-type-swp prm-unit-commutes by metis
  thus ?case by simp
 next
qed auto

end
theory SimplyTyped
imports PreSimplyTyped
begin

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

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

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

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

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

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

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

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

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

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

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

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

lemma *unit-not-snd*:

```
  shows Unit ≠ Snd P
proof(transfer)
  fix P :: 'a ptrm
  show ¬ PUnit ≈ PSnd P
  proof(rule classical)
    assume ¬¬ PUnit ≈ PSnd P
    hence False using unitE by fastforce
    thus ?thesis by blast
  qed
qed

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

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

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

lemma var-not-fst:
  shows Var x ≠ Fst P
```

proof(*transfer*)
  fix x :: $'a$ **and** P
  show $\neg PVar\ x \approx PFst\ P$
  proof(*rule classical*)
    assume $\neg\neg PVar\ x \approx PFst\ P$
    hence *False* **using** *varE* **by** *fastforce*
    thus *?thesis* **by** *blast*
  qed
qed

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

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

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

lemma *app-not-fst*:
  shows $App\ A\ B \neq Fst\ P$
proof(*transfer*)

```
  fix A B P :: 'a ptrm
  show ¬PApp A B ≈ PFst P
  proof(rule classical)
    assume ¬¬PApp A B ≈ PFst P
    hence False using appE by auto
    thus ?thesis by blast
  qed
qed

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

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

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

lemma fn-not-snd:
  shows Fn x T A ≠ Snd P
proof(transfer)
  fix x :: 'a and T A P
```

show $\neg PFn\ x\ T\ A \approx PSnd\ P$
proof(*rule classical*)
assume $\neg\neg PFn\ x\ T\ A \approx PSnd\ P$
hence *False* **using** *fnE* **by** *fastforce*
thus *?thesis* **by** *blast*
qed
qed

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

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

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

lemma *trm-simp*:
shows
$Var\ x = Var\ y \Longrightarrow x = y$
$App\ A\ B = App\ C\ D \Longrightarrow A = C$
$App\ A\ B = App\ C\ D \Longrightarrow B = D$

$Fn\ x\ T\ A = Fn\ y\ S\ B \Longrightarrow$
$(x = y \wedge T = S \wedge A = B) \vee (x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot B)$
$Pair\ A\ B = Pair\ C\ D \Longrightarrow A = C$
$Pair\ A\ B = Pair\ C\ D \Longrightarrow B = D$
$Fst\ P = Fst\ Q \Longrightarrow P = Q$
$Snd\ P = Snd\ Q \Longrightarrow P = Q$
proof −
show $Var\ x = Var\ y \Longrightarrow x = y$ **by** (*transfer*, *insert ptrm.inject varE*, *fastforce*)
show $App\ A\ B = App\ C\ D \Longrightarrow A = C$ **by** (*transfer*, *insert ptrm.inject appE*, *auto*)
show $App\ A\ B = App\ C\ D \Longrightarrow B = D$ **by** (*transfer*, *insert ptrm.inject appE*, *auto*)
show $Pair\ A\ B = Pair\ C\ D \Longrightarrow A = C$ **by** (*transfer*, *insert ptrm.inject pairE*, *auto*)
show $Pair\ A\ B = Pair\ C\ D \Longrightarrow B = D$ **by** (*transfer*, *insert ptrm.inject pairE*, *auto*)
show $Fst\ P = Fst\ Q \Longrightarrow P = Q$ **by** (*transfer*, *insert ptrm.inject fstE*, *auto*)
show $Snd\ P = Snd\ Q \Longrightarrow P = Q$ **by** (*transfer*, *insert ptrm.inject sndE*, *auto*)
show $Fn\ x\ T\ A = Fn\ y\ S\ B \Longrightarrow$
$(x = y \wedge T = S \wedge A = B) \vee (x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot B)$
proof(*transfer'*)
fix $x\ y :: 'a$ **and** $T\ S :: type$ **and** $A\ B :: 'a\ ptrm$
assume $*$: $PFn\ x\ T\ A \approx PFn\ y\ S\ B$
thus $x = y \wedge T = S \wedge A \approx B \vee x \neq y \wedge T = S \wedge x \notin ptrm\text{-}fvs\ B \wedge A \approx [x \leftrightarrow y] \cdot B$
proof(*induction rule*: *fnE*[**where** x=x **and** T=T **and** A=A **and** Y=$PFn\ y\ S\ B$], *metis* $*$)
case (*2 C*)
thus *?case* **by** *simp*
next
case (*3 z C*)
thus *?case* **by** *simp*
next
qed
qed
qed

lemma *fn-eq*:
assumes $x \neq y\ x \notin fvs\ B\ A = [x \leftrightarrow y] \cdot B$
shows $Fn\ x\ T\ A = Fn\ y\ T\ B$
using *assms* **by**(*transfer'*, *metis ptrm-alpha-equiv.fn2*)

lemma *trm-prm-simp*:
shows
$\pi \cdot Unit = Unit$
$\pi \cdot Var\ x = Var\ (\pi\ \$\ x)$
$\pi \cdot App\ A\ B = App\ (\pi \cdot A)\ (\pi \cdot B)$

$\pi \cdot$ *Fn* x T A = *Fn* (π \$ x) T ($\pi \cdot A$)
$\pi \cdot$ *Pair* A B = *Pair* ($\pi \cdot A$) ($\pi \cdot B$)
$\pi \cdot$ *Fst* P = *Fst* ($\pi \cdot P$)
$\pi \cdot$ *Snd* P = *Snd* ($\pi \cdot P$)
apply (*transfer*, *auto simp add*: *ptrm-alpha-equiv-reflexive*)
apply (*transfer'*, *auto simp add*: *ptrm-alpha-equiv-reflexive*)
apply ((*transfer*, *auto simp add*: *ptrm-alpha-equiv-reflexive*)+)
done

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

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

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

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

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

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

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

lemma *trm-prm-agreement-equiv*:
assumes $\bigwedge a.\ a \in$ *ds* $\pi\ \sigma \Longrightarrow a \notin$ *fvs* M
shows $\pi \cdot M = \sigma \cdot M$

using *assms* **by**(*transfer*, *metis ptrm-prm-agreement-equiv*)

lemma *trm-induct*:
  fixes $P :: 'a\ trm \Rightarrow bool$
  assumes
    $P\ Unit$
    $\bigwedge x.\ P\ (Var\ x)$
    $\bigwedge A\ B.\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (App\ A\ B)$
    $\bigwedge x\ T\ A.\ P\ A \Longrightarrow P\ (Fn\ x\ T\ A)$
    $\bigwedge A\ B.\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (Pair\ A\ B)$
    $\bigwedge A.\ P\ A \Longrightarrow P\ (Fst\ A)$
    $\bigwedge A.\ P\ A \Longrightarrow P\ (Snd\ A)$
  shows $P\ M$
proof −
  have $\bigwedge X.\ P\ (abs\text{-}trm\ X)$
  proof(*rule ptrm.induct*)
    show $P\ (abs\text{-}trm\ PUnit)$
      using *assms*(*1*) *Unit.abs-eq* **by** *metis*
    show $P\ (abs\text{-}trm\ (PVar\ x))$ **for** x
      using *assms*(*2*) *Var.abs-eq* **by** *metis*
    show $[\![P\ (abs\text{-}trm\ A);\ P\ (abs\text{-}trm\ B)]\!] \Longrightarrow P\ (abs\text{-}trm\ (PApp\ A\ B))$ **for** $A\ B$
      using *assms*(*3*) *App.abs-eq* **by** *metis*
    show $P\ (abs\text{-}trm\ A) \Longrightarrow P\ (abs\text{-}trm\ (PFn\ x\ T\ A))$ **for** $x\ T\ A$
      using *assms*(*4*) *Fn.abs-eq* **by** *metis*
    show $[\![P\ (abs\text{-}trm\ A);\ P\ (abs\text{-}trm\ B)]\!] \Longrightarrow P\ (abs\text{-}trm\ (PPair\ A\ B))$ **for** $A\ B$
      using *assms*(*5*) *Pair.abs-eq* **by** *metis*
    show $P\ (abs\text{-}trm\ A) \Longrightarrow P\ (abs\text{-}trm\ (PFst\ A))$ **for** A
      using *assms*(*6*) *Fst.abs-eq* **by** *metis*
    show $P\ (abs\text{-}trm\ A) \Longrightarrow P\ (abs\text{-}trm\ (PSnd\ A))$ **for** A
      using *assms*(*7*) *Snd.abs-eq* **by** *metis*
  qed
  thus *?thesis* **using** *trm.abs-induct* **by** *auto*
qed

lemma *trm-cases*:
  assumes
    $M = Unit \Longrightarrow P\ M$
    $\bigwedge x.\ M = Var\ x \Longrightarrow P\ M$
    $\bigwedge A\ B.\ M = App\ A\ B \Longrightarrow P\ M$
    $\bigwedge x\ T\ A.\ M = Fn\ x\ T\ A \Longrightarrow P\ M$
    $\bigwedge A\ B.\ M = Pair\ A\ B \Longrightarrow P\ M$
    $\bigwedge A.\ M = Fst\ A \Longrightarrow P\ M$
    $\bigwedge A.\ M = Snd\ A \Longrightarrow P\ M$
  shows $P\ M$
using *assms* **by**(*induction rule*: *trm-induct*, *auto*)

lemma *trm-depth-induct*:
  assumes
    $P\ Unit$

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

context *fresh* **begin**

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

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

lemma *trm-strong-induct*:
fixes P :: *'a set* ⇒ *'a trm* ⇒ *bool*
assumes
$P\ S\ Unit$
$\bigwedge x.\ P\ S\ (Var\ x)$
$\bigwedge A\ B.\ [\![P\ S\ A;\ P\ S\ B]\!] \Longrightarrow P\ S\ (App\ A\ B)$
$\bigwedge x\ T.\ x \notin S \Longrightarrow (\bigwedge A.\ P\ S\ A \Longrightarrow P\ S\ (Fn\ x\ T\ A))$
$\bigwedge A\ B.\ [\![P\ S\ A;\ P\ S\ B]\!] \Longrightarrow P\ S\ (Pair\ A\ B)$
$\bigwedge A.\ P\ S\ A \Longrightarrow P\ S\ (Fst\ A)$

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

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

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

lemma *trm-strong-cases*:
fixes $P :: {}'a\ set \Rightarrow {}'a\ trm \Rightarrow bool$

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

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

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

inductive *typing* :: $'a\ \mathit{typing\text{-}ctx} \Rightarrow {'a}\ \mathit{trm} \Rightarrow \mathit{type} \Rightarrow \mathit{bool}$ (- ⊢ - : -) **where**
tunit: $\Gamma \vdash \mathit{Unit} : \mathit{TUnit}$
| *tvar*: $\Gamma\ x = \mathit{Some}\ \tau \Longrightarrow \Gamma \vdash \mathit{Var}\ x : \tau$
| *tapp*: $[\![\Gamma \vdash f : (\mathit{TArr}\ \tau\ \sigma); \Gamma \vdash x : \tau]\!] \Longrightarrow \Gamma \vdash \mathit{App}\ f\ x : \sigma$

| *tfn*: $\Gamma(x \mapsto \tau) \vdash A : \sigma \Longrightarrow \Gamma \vdash \mathit{Fn}\ x\ \tau\ A : (\mathit{TArr}\ \tau\ \sigma)$
| *tpair*: $[\![\Gamma \vdash A : \tau; \Gamma \vdash B : \sigma]\!] \Longrightarrow \Gamma \vdash \mathit{Pair}\ A\ B : (\mathit{TPair}\ \tau\ \sigma)$
| *tfst*: $\Gamma \vdash P : (\mathit{TPair}\ \tau\ \sigma) \Longrightarrow \Gamma \vdash \mathit{Fst}\ P : \tau$
| *tsnd*: $\Gamma \vdash P : (\mathit{TPair}\ \tau\ \sigma) \Longrightarrow \Gamma \vdash \mathit{Snd}\ P : \sigma$

lemma *typing-prm*:
  assumes $\Gamma \vdash M : \tau$ $\bigwedge y.\ y \in \mathit{fvs}\ M \Longrightarrow \Gamma\ y = \Delta\ (\pi\ \$\ y)$
  shows $\Delta \vdash \pi \cdot M : \tau$
using *assms* **proof**(*induction arbitrary*: Δ *rule*: *typing.induct*)
  case (*tunit* Γ)
    thus *?case* **using** *typing.tunit trm-prm-simp*(*1*) **by** *metis*
  next
  case (*tvar* Γ x τ)
    thus *?case* **using** *typing.tvar trm-prm-simp*(*2*) *fvs-simp*(*2*) *singletonI* **by** *metis*
  next
  case (*tapp* Γ A τ σ B)
    thus *?case* **using** *typing.tapp trm-prm-simp*(*3*) *fvs-simp*(*3*) *UnCI* **by** *metis*
  next
  case (*tfn* Γ x τ A σ)
    have $y \in \mathit{fvs}\ A \Longrightarrow (\Gamma(x \mapsto \tau))\ y = (\Delta(\pi\ \$\ x \mapsto \tau))\ (\pi\ \$\ y)$ **for** y
    proof(*cases* $y = x$)
      case *True*
        thus *?thesis* **using** *fun-upd-apply* **by** *simp*
      next
      case *False*
        assume $y \in \mathit{fvs}\ A$
        hence $y \in \mathit{fvs}\ (\mathit{Fn}\ x\ \tau\ A)$ **using** *fvs-simp*(*4*) ‹$y \neq x$› *DiffI singletonD* **by** *fastforce*
        hence $\Gamma\ y = \Delta\ (\pi\ \$\ y)$ **using** *tfn.prems* **by** *metis*
        thus *?thesis* **by** (*simp add*: *prm-apply-unequal*)
      next
    qed
    hence $\Delta(\pi\ \$\ x \mapsto \tau) \vdash \pi \cdot A : \sigma$ **using** *tfn.IH* **by** *metis*
    thus *?case* **using** *trm-prm-simp*(*4*) *typing.tfn* **by** *metis*
  next
  case (*tpair* Γ A B)
    thus *?case* **using** *typing.tpair trm-prm-simp*(*5*) *fvs-simp*(*5*) *UnCI* **by** *metis*
  next
  case (*tfst* Γ P τ σ)
    thus *?case* **using** *typing.tfst trm-prm-simp*(*6*) *fvs-simp*(*6*) **by** *metis*
  next
  case (*tsnd* Γ P τ σ)
    thus *?case* **using** *typing.tsnd trm-prm-simp*(*7*) *fvs-simp*(*7*) **by** *metis*
  next
qed

lemma *typing-swp*:
  assumes $\Gamma(a \mapsto \sigma) \vdash M : \tau$ $b \notin \mathit{fvs}\ M$
  shows $\Gamma(b \mapsto \sigma) \vdash [a \leftrightarrow b] \cdot M : \tau$

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

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

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

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

lemma *typing-fnE*:
  assumes $\Gamma \vdash$ *Fn* $x\ T\ A$: ϑ
  shows $\exists \sigma.\ \vartheta = (TArr\ T\ \sigma) \wedge (\Gamma(x \mapsto T) \vdash A : \sigma)$
using *assms* **proof**(*cases*)

```
  case (tfn y S B σ)
    from this consider
      x = y ∧ T = S ∧ A = B | x ≠ y ∧ T = S ∧ x ∉ fvs B ∧ A = [x ↔ y] · B
      using trm-simp(4) by metis
    thus ?thesis proof(cases)
      case 1
        thus ?thesis using tfn by metis
      next
      case 2
        thus ?thesis using tfn typing-swp prm-unit-commutes by metis
      next
    qed
  next
qed (
  metis unit-not-fn,
  metis var-not-fn,
  metis app-not-fn,
  metis fn-not-pair,
  metis fn-not-fst,
  metis fn-not-snd
)

lemma typing-pairE:
  assumes Γ ⊢ Pair A B : ϑ
  shows ∃τ σ. ϑ = (TPair τ σ) ∧ (Γ ⊢ A : τ) ∧ (Γ ⊢ B : σ)
using assms proof(cases)
  case (tpair A τ B σ)
    thus ?thesis using trm-simp(5) trm-simp(6) by blast
  next
qed (
  metis unit-not-pair,
  metis var-not-pair,
  metis app-not-pair,
  metis fn-not-pair,
  metis pair-not-fst,
  metis pair-not-snd
)

lemma typing-fstE:
  assumes Γ ⊢ Fst P : τ
  shows ∃σ. (Γ ⊢ P : (TPair τ σ))
using assms proof(cases)
  case (tfst P σ)
    thus ?thesis using trm-simp(7) by blast
  next
qed (
  metis unit-not-fst,
  metis var-not-fst,
  metis app-not-fst,
```

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

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

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

```
      hence Γ(x ↦ τ, y ↦ σ) ⊢ A : τ' using tfn.IH by metis
      hence Γ(y ↦ σ, x ↦ τ) ⊢ A : τ' using ‹y ≠ x› fun-upd-twist by metis
      thus ?thesis using typing.tfn by metis
    next
  qed
 next
 case (tpair Γ A τ B σ)
  thus ?case using typing.tpair fvs-simp(5) UnCI by metis
 next
 case (tfst Γ P τ σ)
  thus ?case using typing.tfst fvs-simp(6) by metis
 next
 case (tsnd Γ P τ σ)
  thus ?case using typing.tsnd fvs-simp(7) by metis
 next
qed


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

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

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

lemma *infer-unitE*:
  assumes *infer* Γ *Unit* = *Some* τ
  shows τ = *TUnit*
using *assms* **by**(*transfer*, *simp*)

lemma *infer-varE*:
  assumes *infer* Γ (*Var* x) = *Some* τ
  shows Γ x = *Some* τ
using *assms* **by**(*transfer*, *simp*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  have *ptrm-infer-type* Γ $P \neq$ *None*
  proof(*rule classical*, *auto*)
    assume *ptrm-infer-type* Γ $P =$ *None*
    thus *False* **using** H **by** *simp*
  qed
  moreover have *ptrm-infer-type* Γ $P \neq$ *Some TUnit*
  proof(*rule classical*, *auto*)
    assume *ptrm-infer-type* Γ $P =$ *Some TUnit*
    thus *False* **using** H **by** *simp*
  qed
  moreover have *ptrm-infer-type* Γ $P \neq$ *Some* (*TVar* x) **for** x
  proof(*rule classical*, *auto*)
    assume *ptrm-infer-type* Γ $P =$ *Some* (*TVar* x)
    thus *False* **using** H **by** *simp*
  qed
  moreover have *ptrm-infer-type* Γ $P \neq$ *Some* (*TArr* T S) **for** T S
  proof(*rule classical*, *auto*)
    assume *ptrm-infer-type* Γ $P =$ *Some* (*TArr* T S)
    thus *False* **using** H **by** *simp*
  qed
  ultimately obtain T S **where**
    ptrm-infer-type Γ $P =$ *Some* (*TPair* T S)
    using *type.distinct type.exhaust option.exhaust* **by** *metis*
  moreover hence *ptrm-infer-type* Γ (*PSnd* P) $=$ *Some* S **by** *simp*
  ultimately show $\exists\, T\ S.$ *ptrm-infer-type* Γ $P =$ *Some* (*TPair* T S) $\wedge\ \tau = S$
    using H **by** *auto*
qed

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

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

theorem *infer-valid*:
  shows ($\Gamma \vdash M : \tau$) = (*infer* Γ M = *Some* τ)
using *infer-sound infer-complete* **by** *blast*

inductive *substitutes* :: *'a trm* $\Rightarrow$ *'a* $\Rightarrow$ *'a trm* $\Rightarrow$ *'a trm* $\Rightarrow$ *bool* **where**
  unit: *substitutes Unit y M Unit*

| *var1*: $x = y \Longrightarrow$ *substitutes* (*Var x*) *y M M*
| *var2*: $x \neq y \Longrightarrow$ *substitutes* (*Var x*) *y M* (*Var x*)
| *app*: ⟦*substitutes A x M A'*; *substitutes B x M B'*⟧ $\Longrightarrow$ *substitutes* (*App A B*) *x M* (*App A' B'*)
| *fn*: ⟦$x \neq y$; $y \notin$ *fvs M*; *substitutes A x M A'*⟧ $\Longrightarrow$ *substitutes* (*Fn y T A*) *x M* (*Fn y T A'*)
| *pair*: ⟦*substitutes A x M A'*; *substitutes B x M B'*⟧ $\Longrightarrow$ *substitutes* (*Pair A B*) *x M* (*Pair A' B'*)
| *fst*: *substitutes P x M P'* $\Longrightarrow$ *substitutes* (*Fst P*) *x M* (*Fst P'*)
| *snd*: *substitutes P x M P'* $\Longrightarrow$ *substitutes* (*Snd P*) *x M* (*Snd P'*)

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

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

case (*var1 x y M*)
thus *?case* **by** *auto*
next
case (*var2 x y M*)
thus *?case*
using *fvs-simp*(*2*) *Un-subset-iff Un-upper2 insert-Diff-if insert-is-Un singletonD sup-commute*
by *metis*
next
case (*app A x M A′ B B′*)
hence $fvs\ A' \cup fvs\ B' \subseteq (fvs\ A - \{x\} \cup fvs\ M) \cup (fvs\ B - \{x\} \cup fvs\ M)$ **by** *auto*
hence $fvs\ A' \cup fvs\ B' \subseteq (fvs\ A \cup fvs\ B) - \{x\} \cup fvs\ M$ **by** *auto*
thus *?case* **using** *fvs-simp*(*3*) **by** *metis*
next
case (*fn x y M A A′ T*)
hence $fvs\ A' - \{y\} \subseteq fvs\ A - \{y\} - \{x\} \cup fvs\ M$ **by** *auto*
thus *?case* **using** *fvs-simp*(*4*) **by** *metis*
next
case (*pair A x M A′ B B′*)
hence $fvs\ A' \cup fvs\ B' \subseteq (fvs\ A - \{x\} \cup fvs\ M) \cup (fvs\ B - \{x\} \cup fvs\ M)$ **by** *auto*
hence $fvs\ A' \cup fvs\ B' \subseteq (fvs\ A \cup fvs\ B) - \{x\} \cup fvs\ M$ **by** *auto*
thus *?case* **using** *fvs-simp*(*5*) **by** *metis*
next
case (*fst P x M P′*)
thus *?case* **using** *fvs-simp*(*6*) **by** *fastforce*
next
case (*snd P x M P′*)
thus *?case* **using** *fvs-simp*(*7*) **by** *fastforce*
next
qed

inductive-cases *substitutes-unitE′*: *substitutes Unit y M X*
lemma *substitutes-unitE*:
assumes *substitutes Unit y M X*
shows $X = Unit$
by(
rule substitutes-unitE′[**where** $y{=}y$ **and** $M{=}M$ **and** $X{=}X$],
metis assms,
auto simp add: *unit-not-var unit-not-app unit-not-fn unit-not-pair unit-not-fst unit-not-snd*
)

inductive-cases *substitutes-varE′*: *substitutes* (*Var x*) *y M X*
lemma *substitutes-varE*:
assumes *substitutes* (*Var x*) *y M X*
shows $(x = y \wedge M = X) \vee (x \neq y \wedge X = Var\ x)$
by(

rule substitutes-varE′[**where** x=x **and** y=y **and** M=M **and** X=X],
metis assms,
metis unit-not-var,
metis trm-simp(1),
metis trm-simp(1),
auto simp add: *var-not-app var-not-fn var-not-pair var-not-fst var-not-snd*
)

inductive-cases *substitutes-appE′*: *substitutes* (*App A B*) *x M X*
lemma *substitutes-appE*:
assumes *substitutes* (*App A B*) *x M X*
shows $\exists A'\ B'.$ *substitutes* $A\ x\ M\ A' \wedge$ *substitutes* $B\ x\ M\ B' \wedge X = App\ A'\ B'$
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* $y \neq x$ $y \notin$ *fvs M*
shows $\exists A'.$ *substitutes* $A\ x\ M\ A' \wedge X = Fn\ y\ T\ A'$
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 $y = z \wedge T = S \wedge A = B \mid y \neq z \wedge T = S \wedge y \notin fvs\ B \wedge A = [y \leftrightarrow z] \cdot B$
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 $y \neq z$ $T = S$ $y \notin fvs\ B$ $A = [y \leftrightarrow z] \cdot B$ **by** *auto*
have *substitutes* $([y \leftrightarrow z] \cdot B)\ ([y \leftrightarrow z]\ \$\ x)\ ([y \leftrightarrow z] \cdot M)\ ([y \leftrightarrow z] \cdot B')$
using *substitutes-prm* ‹*substitutes B x M B′*› **by** *metis*
hence *substitutes* $A\ ([y \leftrightarrow z]\ \$\ x)\ ([y \leftrightarrow z] \cdot M)\ ([y \leftrightarrow z] \cdot B')$
using ‹$A = [y \leftrightarrow z] \cdot B$› **by** *metis*
hence *substitutes* $A\ x\ ([y \leftrightarrow z] \cdot M)\ ([y \leftrightarrow z] \cdot B')$
using ‹$y \neq x$› ‹$x \neq z$› *prm-unit-inaction* **by** *metis*
hence $*$: *substitutes* $A\ x\ M\ ([y \leftrightarrow z] \cdot B')$
using ‹$y \notin fvs\ M$› ‹$z \notin fvs\ M$› *trm-prm-unit-inaction* **by** *metis*

have $y \notin fvs\ B'$

using
$substitutes\text{-}fvs$ ‹$substitutes\ B\ x\ M\ B'$› ‹$y \notin fvs\ B$› ‹$y \notin fvs\ M$›
$Diff\text{-}subset\ UnE\ rev\text{-}subsetD$
by *blast*
hence $X = Fn\ y\ T\ ([y \leftrightarrow z] \cdot B')$
using ‹$X = Fn\ z\ S\ B'$› ‹$y \neq z$› ‹$T = S$› *fn-eq*
by *metis*

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

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

inductive-cases *substitutes-fstE'*: $substitutes\ (Fst\ P)\ x\ M\ X$
lemma *substitutes-fstE*:
assumes $substitutes\ (Fst\ P)\ x\ M\ X$
shows $\exists\, P'.\ substitutes\ P\ x\ M\ P' \wedge X = Fst\ P'$
proof(*cases rule*: *substitutes-fstE'*[**where** P=P **and** x=x **and** M=M **and** X=X])
case ($8\ P\ P'$)

```
    thus ?thesis using trm-simp(7) by blast
  next
qed (
  metis assms,
  metis unit-not-fst,
  metis var-not-fst,
  metis var-not-fst,
  metis app-not-fst,
  metis fn-not-fst,
  metis pair-not-fst,
  metis fst-not-snd
)

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

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

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

```
      next
      case 2
        obtain X where X = (Var y) by auto
        hence substitutes (Var y) x M X using ‹x ≠ y› substitutes.var2 by metis
        thus ?thesis by auto
      next
    qed
  next
  case (3 A B)
    from this obtain A′ B′ where A′: substitutes A x M A′ and B′: substitutes B
x M B′ by auto
    obtain X where X = App A′ B′ by auto
    hence substitutes (App A B) x M X using A′ B′ substitutes.app by metis
    thus ?case by auto
  next
  case (4 y T A)
    from this obtain A′ where A′: substitutes A x M A′ by auto
    from ‹y ∉ ({x} ∪ fvs M)› have y ≠ x y ∉ fvs M by auto
    obtain X where X = Fn y T A′ by auto
    hence substitutes (Fn y T A) x M X using substitutes.fn ‹y ≠ x› ‹y ∉ fvs M›
A′ by metis
    thus ?case by auto
  next
  case (5 A B)
    from this obtain A′ B′ where substitutes A x M A′ substitutes B x M B′ by
auto
    from this obtain X where X = Pair A′ B′ by auto
    hence substitutes (Pair A B) x M X
      using substitutes.pair ‹substitutes A x M A′› ‹substitutes B x M B′›
      by metis
    thus ?case by auto
  next
  case (6 P)
    from this obtain P′ where substitutes P x M P′ by auto
    from this obtain X where X = Fst P′ by auto
    hence substitutes (Fst P) x M X using substitutes.fst ‹substitutes P x M P′›
by metis
    thus ?case by auto
  next
  case (7 P)
    from this obtain P′ where substitutes P x M P′ by auto
    from this obtain X where X = Snd P′ by auto
    hence substitutes (Snd P) x M X using substitutes.snd ‹substitutes P x M P′›
by metis
    thus ?case by auto
  next
qed

lemma substitutes-unique:
```

assumes *substitutes A x M B substitutes A x M C*
shows $B = C$
using *assms* **proof**(*induction A arbitrary*: *B C rule*: *trm-strong-induct*[**where** $S=\{x\} \cup$ *fvs M*])
show *finite* ($\{x\} \cup$ *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** $y \notin$ *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 $\exists!\ X.$ *substitutes A x M X*
using *substitutes-total substitutes-unique* **by** *metis*

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

lemma *subst-simp-unit*:
shows *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 $(\mathit{Var}\ x)[y ::= M] = \mathit{Var}\ x$
unfolding *subst-def* **by**(
rule,
metis substitutes.var2 assms,
metis substitutes-function substitutes.var2 assms
)

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

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

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

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

lemma *subst-simp-pair*:
shows $(\mathit{Pair}\ A\ B)[x ::= M] = \mathit{Pair}\ (A[x ::= M])\ (B[x ::= M])$
unfolding *subst-def* **proof**
obtain $A'\ B'$ **where** A': $A' = (A[x ::= M])$ **and** B': $B' = (B[x ::= M])$ **by** *auto*
hence *substitutes* $A\ x\ M\ A'$ *substitutes* $B\ x\ M\ B'$

unfolding *subst-def* **using** *substitutes-function theI* **by** *metis+*
hence *substitutes* (*Pair A B*) *x M* (*Pair A′ B′*) **using** *substitutes.pair* **by** *metis*
thus ∗: *substitutes* (*Pair A B*) *x M* (*Pair* (*THE X*. *substitutes A x M X*) (*THE X*. *substitutes B x M X*))
using *A′ B′* **unfolding** *subst-def* **by** *metis*

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

lemma *subst-simp-fst*:
shows (*Fst P*)[*x* ::= *M*] = *Fst* (*P*[*x* ::= *M*])
unfolding *subst-def* **proof**
obtain *P′* **where** *P′*: *P′* = (*P*[*x* ::= *M*]) **by** *auto*
hence *substitutes P x M P′* **unfolding** *subst-def* **using** *substitutes-function theI* **by** *metis*
hence *substitutes* (*Fst P*) *x M* (*Fst P′*) **using** *substitutes.fst* **by** *metis*
thus ∗: *substitutes* (*Fst P*) *x M* (*Fst* (*THE X*. *substitutes P x M X*))
using *P′* **unfolding** *subst-def* **by** *metis*

fix *X*
assume *substitutes* (*Fst P*) *x M X*
thus *X* = *Fst* (*THE X*. *substitutes P x M X*) **using** *substitutes-function* ∗ **by** *metis*
qed

lemma *subst-simp-snd*:
shows (*Snd P*)[*x* ::= *M*] = *Snd* (*P*[*x* ::= *M*])
unfolding *subst-def* **proof**
obtain *P′* **where** *P′*: *P′* = (*P*[*x* ::= *M*]) **by** *auto*
hence *substitutes P x M P′* **unfolding** *subst-def* **using** *substitutes-function theI* **by** *metis*
hence *substitutes* (*Snd P*) *x M* (*Snd P′*) **using** *substitutes.snd* **by** *metis*
thus ∗: *substitutes* (*Snd P*) *x M* (*Snd* (*THE X*. *substitutes P x M X*))
using *P′* **unfolding** *subst-def* **by** *metis*

fix *X*
assume *substitutes* (*Snd P*) *x M X*
thus *X* = *Snd* (*THE X*. *substitutes P x M X*) **using** *substitutes-function* ∗ **by** *metis*
qed

lemma *subst-prm*:
shows (π · (*M*[*z* ::= *N*])) = ((π · *M*)[π $ *z* ::= π · *N*])
unfolding *subst-def* **using** *substitutes-prm substitutes-function the1-equality* **by** (*metis* (*full-types*))

lemma *subst-fvs*:
  shows *fvs* ($M[z ::= N]$) $\subseteq$ (*fvs* M $-$ $\{z\}$) $\cup$ *fvs* N
unfolding *subst-def* **using** *substitutes-fvs substitutes-function theI2* **by** *metis*

lemma *subst-free*:
  assumes $y \notin$ *fvs* M
  shows $M[y ::= N] = M$
using *assms* **proof**(*induction M rule*: *trm-strong-induct*[**where** $S=\{y\} \cup$ *fvs* N])
  show *finite* ($\{y\} \cup$ *fvs* N) **using** *fvs-finite* **by** *auto*

  case *1*
    thus *?case* **using** *subst-simp-unit* **by** *metis*
  next
  case (*2 x*)
    thus *?case* **using** *subst-simp-var2* **by** (*simp add*: *fvs-simp*)
  next
  case (*3 A B*)
    thus *?case* **using** *subst-simp-app* **by** (*simp add*: *fvs-simp*)
  next
  case (*4 x T A*)
    hence $y \neq x$ $x \notin$ *fvs* N **by** *auto*

    have $y \notin$ *fvs* A $-$ $\{x\}$ **using** ‹$y \neq x$› ‹$y \notin$ *fvs* (*Fn* x T A)› *fvs-simp*(*4*) **by** *metis*
    hence $y \notin$ *fvs* A **using** ‹$y \neq x$› **by** *auto*
    hence $A[y ::= N] = A$ **using** *4.IH* **by** *blast*
    thus *?case* **using** ‹$y \neq x$› ‹$y \notin$ *fvs* A› ‹$x \notin$ *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 $y \notin$ *fvs* A
  shows ($[y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M]$)
using *assms* **proof**(*induction A rule*: *trm-strong-induct*[**where** $S=\{y,\ z\} \cup$ *fvs* M])
  show *finite* ($\{y,\ z\} \cup$ *fvs* M) **using** *fvs-finite* **by** *auto*
  next

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

next
case (*2 x*)
hence $y \neq x$ **using** *fvs-simp*(*2*) **by** *force*
consider $x = z \mid x \neq z$ **by** *auto*
thus *?case* **proof**(*cases*)
case *1*
thus *?thesis* **using** *subst-simp-var1 trm-prm-simp*(*2*) *prm-unit-action prm-unit-commutes* **by** *metis*
next
case *2*
thus *?thesis* **using** *subst-simp-var2 trm-prm-simp*(*2*) *prm-unit-inaction* ‹$y \neq x$› **by** *metis*
next
qed
next
case (*3 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 *?case* **using** *3.IH subst-simp-app trm-prm-simp*(*3*) **by** *metis*
next
case (*4 x T A*)
hence $x \neq y$ $x \neq z$ $x \notin$ *fvs M* **by** *auto*
hence $y \notin$ *fvs A* **using** ‹$y \notin$ *fvs* (*Fn x T A*)› *fvs-simp*(*4*) **by** *force*
hence $*$: $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M])$ **using** *4.IH* **by** *metis*

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

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

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

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

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

using *subst-simp-fn* ‹$x \neq y$› ‹$x \neq z$› ‹$x \notin$ *fvs* $(N[z ::= L])$› ‹$x \notin$ *fvs* L› **by** *metis*
finally show *?case*.
next
case (*5 A B*)
thus *?case* **using** *subst-simp-pair* **by** *metis*
next
case (*6 P*)
thus *?case* **using** *subst-simp-fst* **by** *metis*
next
case (*7 P*)
thus *?case* **using** *subst-simp-snd* **by** *metis*
next
qed

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

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

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

from ‹$\Gamma(z \mapsto \tau) \vdash$ *App* A $B : \sigma$› **obtain** σ' **where**
$\Gamma(z \mapsto \tau) \vdash A :$ (*TArr* σ' σ)

$\Gamma(z \mapsto \tau) \vdash B : \sigma'$
using *typing-appE* **by** *metis*
hence
$\Gamma \vdash (A[z ::= N]) : (TArr\ \sigma'\ \sigma)$
$\Gamma \vdash (B[z ::= N]) : \sigma'$
using *3* $*$ **by** *metis+*
hence $\Gamma \vdash App\ (A[z ::= N])\ (B[z ::= N]) : \sigma$ **using** *typing.tapp* **by** *metis*
thus *?case* **using** *subst-simp-app* **by** *metis*
next
case $(4\ x\ T\ A)$
hence $x \neq z\ x \notin fvs\ N$ **by** *auto*
hence $*$: $\Gamma(x \mapsto T) \vdash N : \tau$ **using** *typing-weaken 4* **by** *metis*
have $**$: $depth\ A < depth\ (Fn\ x\ T\ A)$ **using** *depth-fn*.

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

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

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

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

next
case (*snd1 P P*′)
thus *?case* **using** *fvs-simp*(*7*) **by** *metis*
next
case (*snd2 A B*)
thus *?case* **using** *fvs-simp*(*5*, *7*) **by** *force*
next
qed

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

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

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

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

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

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

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

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

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

metis fst-not-snd
)

lemma *beta-reduction-left-sndE*:
assumes $(Snd\ P) \rightarrow\beta\ M'$
shows $(\exists P'. (P \rightarrow\beta\ P') \wedge M' = (Snd\ P')) \vee (\exists A\ B. P = Pair\ A\ B \wedge M' = B)$
using *assms* **proof**(*cases*)
case (*snd1 P P'*)
thus *?thesis* **using** *trm-simp*(*8*) **by** *blast*
next
case (*snd2 A*)
thus *?thesis* **using** *trm-simp*(*8*) **by** *blast*
next
qed (
metis app-not-snd,
metis app-not-snd,
metis app-not-snd,
metis fn-not-snd,
metis pair-not-snd,
metis pair-not-snd,
metis fst-not-snd,
metis fst-not-snd
)

lemma *preservation'*:
assumes $\Gamma \vdash M : \tau$ **and** $M \rightarrow\beta\ M'$
shows $\Gamma \vdash M' : \tau$
using *assms* **proof**(*induction arbitrary*: M' *rule*: *typing.induct*)
case (*tunit* Γ)
thus *?case* **using** *beta-reduction-left-unitE* **by** *metis*
next
case (*tvar* $\Gamma\ x\ \tau$)
thus *?case* **using** *beta-reduction-left-varE* **by** *metis*
next
case (*tapp* $\Gamma\ A\ \tau\ \sigma\ B\ M'$)
from ‹$(App\ A\ B) \rightarrow\beta\ M'$› **consider**
$(\exists x\ T\ X. A = (Fn\ x\ T\ X) \wedge M' = (X[x ::= B]))$ |
$(\exists A'. (A \rightarrow\beta\ A') \wedge M' = App\ A'\ B)$ |
$(\exists B'. (B \rightarrow\beta\ B') \wedge M' = App\ A\ B')$ **using** *beta-reduction-left-appE* **by** *metis*

thus *?case* **proof**(*cases*)
case *1*
from *this* **obtain** $x\ T\ X$ **where** $A = Fn\ x\ T\ X$ **and** $*$: $M' = (X[x ::= B])$
by *auto*

have $\Gamma(x \mapsto \tau) \vdash X : \sigma$ **using** *typing-fnE* ‹$\Gamma \vdash A : (TArr\ \tau\ \sigma)$› ‹$A = Fn\ x$
$T\ X$› *type.inject*
by *blast*
hence $\Gamma \vdash (X[x ::= B]) : \sigma$ **using** *typing-subst* ‹$\Gamma \vdash B : \tau$› **by** *metis*

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

```
      from this obtain A B where P = Pair A B and M′ = A by auto
      thus ?thesis using ‹Γ ⊢ P : (TPair τ σ)› typing-pairE by blast
    next
  qed
next
case (tsnd Γ P τ σ)
  from this consider
    ∃ P′. (P →β P′) ∧ M′ = Snd P′
  | ∃ A B. P = Pair A B ∧ M′ = B using beta-reduction-left-sndE by metis
  thus ?case proof(cases)
    case 1
      from this obtain P′ where P →β P′ and M′ = Snd P′ by auto
      thus ?thesis using tsnd typing.tsnd by metis
    next
    case 2
      from this obtain A B where P = Pair A B and M′ = B by auto
      thus ?thesis using ‹Γ ⊢ P : (TPair τ σ)› typing-pairE by blast
    next
  qed
next
qed

inductive NF :: ′a trm ⇒ bool where
  unit: NF Unit
| var: NF (Var x)
| app: ⟦A ≠ Fn x T A′; NF A; NF B⟧ ⟹ NF (App A B)
| fn: NF A ⟹ NF (Fn x T A)
| pair: ⟦NF A; NF B⟧ ⟹ NF (Pair A B)
| fst: ⟦P ≠ Pair A B; NF P⟧ ⟹ NF (Fst P)
| snd: ⟦P ≠ Pair A B; NF P⟧ ⟹ NF (Snd P)

theorem progress:
  assumes Γ ⊢ M : τ
  shows NF M ∨ (∃ M′. (M →β M′))
using assms proof(induction M arbitrary: Γ τ rule: trm-induct)
  case 1
    thus ?case using NF.unit by metis
  next
  case (2 x)
    thus ?case using NF.var by metis
  next
  case (3 A B)
    from ‹Γ ⊢ App A B : τ› obtain σ
      where Γ ⊢ A : (TArr σ τ) and Γ ⊢ B : σ
      using typing-appE by metis
    hence A: NF A ∨ (∃ A′. (A →β A′)) and B: NF B ∨ (∃ B′. (B →β B′))
      using 3.IH by auto

    consider NF B | ∃ B′. (B →β B′) using B by auto
```

```
    thus ?case proof(cases)
      case 1
        consider NF A | ∃ A′. (A →β A′) using A by auto
        thus ?thesis proof(cases)
          case 1
            consider ∃ x T A′. A = Fn x T A′ | ∄ x T A′. A = Fn x T A′ by auto
            thus ?thesis proof(cases)
              case 1
                from this obtain x T A′ where A = Fn x T A′ by auto
                  hence (App A B) →β (A′[x ::= B]) using beta-reduction.beta by
metis
                thus ?thesis by blast
              next
              case 2
                thus ?thesis using ‹NF A› ‹NF B› NF.app by metis
              next
            qed
          next
          case 2
            thus ?thesis using beta-reduction.app1 by metis
          next
        qed
      next
      case 2
        thus ?thesis using beta-reduction.app2 by metis
      next
    qed
  next
  case (4 x T A)
    from ‹Γ ⊢ Fn x T A : τ› obtain σ
      where τ = TArr T σ and Γ(x ↦ T) ⊢ A : σ
      using typing-fnE by metis
    from ‹Γ(x ↦ T) ⊢ A : σ› consider NF A | ∃ A′. (A →β A′)
      using 4.IH by metis

    thus ?case proof(cases)
      case 1
        thus ?thesis using NF.fn by metis
      next
      case 2
        from this obtain A′ where A →β A′ by auto
        obtain M′ where M′ = Fn x T A′ by auto
        hence (Fn x T A) →β M′ using ‹A →β A′› beta-reduction.fn by metis
        thus ?thesis by auto
      next
    qed
  next
  case (5 A B)
   thus ?case using typing-pairE beta-reduction.pair1 beta-reduction.pair2 NF.pair
```

```
by meson
  next
  case (6 P)
    from this consider NF P | ∃ P'. (P →β P') using typing-fstE by metis
    thus ?case proof(cases)
      case 1
        consider ∃ A B. P = Pair A B | ∄ 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) →β A using beta-reduction.fst2 by metis
            thus ?thesis by auto
          next
          case 2
            thus ?thesis using ‹NF P› NF.fst by metis
          next
        qed
      next
      case 2
        thus ?thesis using beta-reduction.fst1 by metis
      next
    qed
  next
  case (7 P)
    from this consider NF P | ∃ P'. (P →β P') using typing-sndE by metis
    thus ?case proof(cases)
      case 1
        consider ∃ A B. P = Pair A B | ∄ 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) →β B using beta-reduction.snd2 by metis
            thus ?thesis by auto
          next
          case 2
            thus ?thesis using ‹NF P› NF.snd by metis
          next
        qed
      next
      case 2
        thus ?thesis using beta-reduction.snd1 by metis
      next
    qed
  next
qed

inductive beta-reduces :: 'a trm ⇒ 'a trm ⇒ bool (- →β* -) where
  reflexive: M →β* M
| transitive: ⟦M →β* M'; M' →β M''⟧ ⟹ M →β* 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** ‹*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 →β* A′
  shows (Fn x T A) →β* (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 →β* A′
  shows (Pair A B) →β* (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 →β* B′
  shows (Pair A B) →β* (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 →β* P′
  shows (Fst P) →β* (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 $P \to\beta^* \ P'$
  shows $(Snd\ P) \to\beta^* \ (Snd\ P')$
using *assms* **proof**(*induction*)
  case (*reflexive M*)
    thus *?case* **using** *beta-reduces.reflexive*.
  next
  case (*transitive* $M\ M'\ M''$)
    thus *?case* **using** *beta-reduces.transitive beta-reduction.snd1* **by** *metis*
  next
qed

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

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

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

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 $*$:*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 *?case* **using** *fvs-simp*(*5*, *7*) **by** *force*
next
qed

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

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

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

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

inductive-cases *parallel-reduction-redexE'*: $(App\ (Fn\ x\ T\ A)\ B) >> X$
lemma *parallel-reduction-redexE*:
assumes $(App\ (Fn\ x\ T\ A)\ B) >> X$
shows
$(X = App\ (Fn\ x\ T\ A)\ B) \vee$
$(\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge X = (A'[x ::= B'])) \vee$
$(\exists A'\ B'.\ ((Fn\ x\ T\ A) >> (Fn\ x\ T\ A')) \wedge (B >> B') \wedge X = (App\ (Fn\ x\ T\ A')$
$B'))$

using *assms* **proof**(*induction rule*: *parallel-reduction-redexE'*[**where** x=x **and** T=T **and** A=A **and** B=B **and** X=X])
case $(5\ C\ C'\ D\ D')$
from ‹$App\ (Fn\ x\ T\ A)\ B = App\ C\ D$› **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**
$x = y \wedge T = S \wedge A = C$
| $x \neq y \wedge T = S \wedge A = [x \leftrightarrow y] \cdot C \wedge x \notin fvs\ C$
using *trm-simp*(*4*) **by** *metis*
thus *?case* **proof**(*cases*)
case *1*
thus *?thesis* **using** ‹$C >> C'$› ‹$X = (C'[y ::= D'])$› ‹$D >> D'$› B **by** *metis*
next
case *2*
hence $x \neq y$ $T = S$ **and** A: $A = [x \leftrightarrow y] \cdot C$ $x \notin fvs\ C$ **by** *auto*
have $x \notin fvs\ C'$ **using** *parallel-reduction-fvs* ‹$x \notin fvs\ C$› ‹$C >> C'$› **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-swp* ‹$x \notin fvs\ C'$› **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** $\bigwedge x\ T\ A'.\ A \neq Fn\ x\ T\ A'$
shows $\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge X = (App\ A'\ B')$
using *assms* **proof**(*induction rule*: *parallel-reduction-nonredexE'*[**where** A=A **and** B=B **and** X=X])
case $(5\ C\ C'\ D\ D')$
hence $A = C\ B = D$ **using** *trm-simp*(*2*, *3*) **by** *auto*
thus *?case* **using** ‹$C >> C'$› ‹$D >> D'$› ‹$X = App\ C'\ D'$› **by** *metis*
next
qed (
metis assms(*1*),
metis parallel-reduction.refl,
metis trm-simp(*2*, *3*) *assms*(*2*),
metis app-not-fn,
metis app-not-pair,
metis app-not-fst,
metis app-not-fst,
metis app-not-snd,
metis app-not-snd
)

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

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

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

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

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

lemma parallel-reduction-subst-inner:
  assumes M >> M'
  shows X[z ::= M] >> (X[z ::= M'])
using assms proof(induction X rule: trm-strong-induct[where S={z} ∪ fvs M ∪
fvs M'])
  show finite ({z} ∪ fvs M ∪ fvs M') using fvs-finite by auto

  case 1
    thus ?case using subst-simp-unit parallel-reduction.refl by metis
  next
  case (2 x)
    thus ?case by(cases x = z, metis subst-simp-var1, metis subst-simp-var2 par-
allel-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 ≠ z x ∉ fvs M x ∉ fvs M' by auto
    thus ?case using 4 subst-simp-fn parallel-reduction.eta by metis
  next
  case (5 A B)
    thus ?case using subst-simp-pair parallel-reduction.pair by metis
  next
  case (6 P)
    thus ?case using subst-simp-fst parallel-reduction.fst1 by metis
  next
  case (7 P)
    thus ?case using subst-simp-snd parallel-reduction.snd1 by metis
  next
qed

lemma parallel-reduction-subst:
  assumes X >> 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} ∪ fvs M ∪ fvs M'])
  show finite ({z} ∪ fvs M ∪ fvs M') using fvs-finite by auto
  next

  case 1
```

hence $X' = Unit$ **using** *parallel-reduction-unitE* **by** *metis*
thus *?case* **using** *parallel-reduction.refl subst-simp-unit* **by** *metis*
next
case $(2\ x)$
hence $X' = Var\ x$ **using** *parallel-reduction-varE* **by** *metis*
thus *?case* **using** *parallel-reduction-subst-inner* ‹$M >> M'$› **by** *metis*
next
case $(3\ C\ D)$
consider $\exists x\ T\ A.\ C = Fn\ x\ T\ A \mid \nexists x\ T\ A.\ C = Fn\ x\ T\ A$ **by** *metis*
thus *?case* **proof**(*cases*)
case *1*
from *this* **obtain** $x\ T\ A$ **where** C: $C = Fn\ x\ T\ A$ **by** *auto*
have $depth\ C < depth\ (App\ C\ D)$ $depth\ D < depth\ (App\ C\ D)$
using *depth-app* **by** *auto*

consider
$X' = App\ (Fn\ x\ T\ A)\ D$
$\mid \exists A'\ D'.\ ((Fn\ x\ T\ A) >> (Fn\ x\ T\ A')) \wedge (D >> D') \wedge X' = (App\ (Fn\ x$
$T\ A')\ D')$
$\mid \exists A'\ D'.\ (A >> A') \wedge (D >> D') \wedge X' = (A'[x ::= D'])$
using *parallel-reduction-redexE* ‹$(App\ C\ D) >> X'$› C **by** *metis*
thus *?thesis* **proof**(*cases*)
case *1*
thus *?thesis* **using** *parallel-reduction-subst-inner* ‹$M >> M'$› C **by** *metis*
next
case *2*
from *this* **obtain** $A'\ D'$
where $(Fn\ x\ T\ A) >> (Fn\ x\ T\ A')$ $D >> D'$ **and** X': $X' = App\ (Fn$
$x\ T\ A')\ D'$
by *auto*
have $*$: $((Fn\ x\ T\ A)[z ::= M]) >> ((Fn\ x\ T\ A')[z ::= M'])$
using *3.IH* ‹$depth\ C < depth\ (App\ C\ D)$› C ‹$(Fn\ x\ T\ A) >> (Fn\ x\ T$
$A')$› ‹$M >> M'$›
by *metis*
have $**$: $(D[z ::= M]) >> (D'[z ::= M'])$
using *3.IH* ‹$depth\ D < depth\ (App\ C\ D)$› ‹$D >> D'$› ‹$M >> M'$›
by *metis*

have $(App\ C\ D)[z ::= M] = App\ ((Fn\ x\ T\ A)[z ::= M])\ (D[z ::= M])$
using *subst-simp-app* C **by** *metis*
moreover have $\ldots >> (App\ ((Fn\ x\ T\ A')[z ::= M'])\ (D'[z ::= M']))$
using $*$ $**$ *parallel-reduction.app* **by** *metis*
moreover have $\ldots = ((App\ (Fn\ x\ T\ A')\ D')[z ::= M'])$
using *subst-simp-app* **by** *metis*
moreover have $\ldots = (X'[z ::= M'])$
using X' **by** *metis*
ultimately show *?thesis* **by** *metis*
next
case *3*

from *this* **obtain** A' D' **where** $A >> A'$ $D >> D'$ **and** X': $X' = (A'[x ::= D'])$
by *auto*

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

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

have (*App* C D)$[z ::= M] =$ (*App* ((*Fn* y T ($[y \leftrightarrow x] \cdot A$))$[z ::= M]$) ($D[z ::= M]$))
using C *subst-simp-app* **by** *metis*
moreover have ... = (*App* (*Fn* y T (($[y \leftrightarrow x] \cdot A)[z ::= M]$)) ($D[z ::= M]$))
using ‹$y \neq z$› ‹$y \notin$ *fvs* M› *subst-simp-fn* **by** *metis*
moreover have ... $>> (([y \leftrightarrow x] \cdot A')[z ::= M'][y ::= D'[z ::= M']])$
using *parallel-reduction.beta* $*$ $**$ **by** *metis*
moreover have ... $= (([y \leftrightarrow x] \cdot A')[y ::= D'][z ::= M'])$
using *barendregt* ‹$y \neq z$› ‹$y \notin$ *fvs* M'› **by** *metis*
moreover have ... $= (A'[x ::= D'][z ::= M'])$
using *subst-swp* ‹$y \notin$ *fvs* 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': $X' =$ *App* C' D'
using *parallel-reduction-nonredexE* ‹(*App* C D) $>> X'$› **by** *metis*

have *depth* $C <$ *depth* (*App* C D) *depth* $D <$ *depth* (*App* C D)
using *depth-app* **by** *auto*
hence $*$: $(C[z ::= M]) >> (C'[z ::= M'])$ **and** $**$: $(D[z ::= M]) >> (D'[z ::= M'])$
using *3.IH* ‹$C >> C'$› ‹$D >> D'$› ‹$M >> M'$› **by** *metis+*

    have $(App\ C\ D)[z ::= M] = App\ (C[z ::= M])\ (D[z ::= M])$
      using *subst-simp-app* **by** *metis*
    moreover have ... $>> (App\ (C'[z ::= M'])\ (D'[z ::= M']))$
      using *parallel-reduction.app* $*$ $**$ **by** *metis*
    moreover have ... $= ((App\ C'\ D')[z ::= M'])$
      using *subst-simp-app* **by** *metis*
    moreover have ... $= (X'[z ::= M'])$ **using** X' **by** *metis*
    ultimately show *?thesis* **by** *metis*
  next
 qed
next
case $(4\ x\ T\ A)$
 hence $x \neq z$ $x \notin fvs\ M$ $x \notin fvs\ M'$
  by *auto*

 from ‹$(Fn\ x\ T\ A) >> X'$› **consider**
  $X' = Fn\ x\ T\ A$
 | $\exists A'.\ (A >> A') \wedge X' = Fn\ x\ T\ A'$ **using** *parallel-reduction-fnE* **by** *metis*
 thus *?case* **proof**(*cases*)
  case *1*
    thus *?thesis* **using** *parallel-reduction-subst-inner* ‹$M >> M'$› **by** *metis*
  next
  case *2*
    from *this* **obtain** A' **where** $A >> A'$ **and** X': $X' = Fn\ x\ T\ A'$ **by** *auto*

    hence $*$: $(A[z ::= M]) >> (A'[z ::= M'])$
      using *4.IH depth-fn* ‹$A >> A'$› ‹$M >> M'$› **by** *metis*

    have $((Fn\ x\ T\ A)[z ::= M]) = (Fn\ x\ T\ (A[z ::= M]))$
      using *subst-simp-fn* ‹$x \neq z$› ‹$x \notin fvs\ M$› **by** *metis*
    moreover have ... $>> (Fn\ x\ T\ (A'[z ::= M']))$
      using *parallel-reduction.eta* $*$ **by** *metis*
    moreover have ... $= ((Fn\ x\ T\ A')[z ::= M'])$
      using *subst-simp-fn* ‹$x \neq z$› ‹$x \notin fvs\ M'$› **by** *metis*
    moreover have ... $= (X'[z ::= M'])$
      using X' **by** *metis*
    ultimately show *?thesis* **by** *metis*
  next
 qed
next
case $(5\ A\ B)$
 from ‹$(Pair\ A\ B) >> X'$› **consider**
  $X' = Pair\ A\ B$
 | $\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge X' = Pair\ A'\ B'$
  using *parallel-reduction-pairE* **by** *metis*
 thus *?case* **proof**(*cases*)
  case *1*
    thus *?thesis* **using** *parallel-reduction-subst-inner* ‹$M >> M'$› **by** *metis*

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

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

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

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

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

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

using $6.IH$ ‹$A >> A'$› ‹$M >> M'$› **by** *metis*

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

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

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

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

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

next
qed
next
qed

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

lemma *not-fn-prm*:
assumes $\bigwedge x\ T\ M.\ A \neq Fn\ x\ T\ M$
shows $\pi \cdot A \neq Fn\ x\ T\ M$
proof(*rule classical*)
fix *x T M*
obtain π' **where** $*$: $\pi' = prm\text{-}inv\ \pi$ **by** *auto*
assume $\neg(\pi \cdot A \neq Fn\ x\ T\ M)$
hence $\pi \cdot A = Fn\ x\ T\ M$ **by** *blast*
hence $\pi' \cdot (\pi \cdot A) = \pi' \cdot Fn\ x\ T\ M$ **by** *fastforce*
hence $(\pi' \diamond \pi) \cdot A = \pi' \cdot Fn\ x\ T\ M$
using *trm-prm-apply-compose* **by** *metis*
hence $A = \pi' \cdot Fn\ x\ T\ M$
using $*$ *prm-inv-compose trm-prm-apply-id* **by** *metis*
hence $A = Fn\ (\pi'\ \$\ x)\ T\ (\pi' \cdot M)$ **using** *trm-prm-simp*(*4*) **by** *metis*
hence *False* **using** *assms* **by** *blast*
thus *?thesis* **by** *blast*
qed

lemma *not-pair-prm*:
assumes $\bigwedge A\ B.\ P \neq Pair\ A\ B$
shows $\pi \cdot P \neq Pair\ A\ B$
proof(*rule classical*)
fix *A B*
obtain π' **where** $*$: $\pi' = prm\text{-}inv\ \pi$ **by** *auto*
assume $\neg(\pi \cdot P \neq Pair\ A\ B)$
hence $\pi \cdot P = Pair\ A\ B$ **by** *blast*
hence $\pi' \cdot \pi \cdot P = \pi' \cdot (Pair\ A\ B)$ **by** *fastforce*
hence $(\pi' \diamond \pi) \cdot P = \pi' \cdot (Pair\ A\ B)$
using *trm-prm-apply-compose* **by** *metis*
hence $P = \pi' \cdot (Pair\ A\ B)$
using $*$ *prm-inv-compose trm-prm-apply-id* **by** *metis*

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

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

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

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

lemma *complete-development-fnE*:
  assumes (*Fn x T A*) >>> *X*
  shows $\exists A'.\ (A >>> A') \wedge X = Fn\ x\ T\ A'$
using *assms* **proof**(*cases*)
  case (*eta B B′ y S*)
    hence $T = S$ **using** *trm-simp*(*4*) **by** *metis*
    from *eta* **consider** $x = y \wedge A = B \mid x \neq y \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot B$
      using *trm-simp*(*4*) **by** *metis*
    thus *?thesis* **proof**(*cases*)
      case *1*
        hence $x = y$ **and** $A = B$ **by** *auto*

obtain A' **where** $A' = B'$ **by** *auto*
hence $A >>> A'$ **and** $X = Fn\ x\ T\ A'$ **using** *eta* ‹$A = B$› ‹$x = y$› ‹$T = S$› **by** *auto*
thus *?thesis* **by** *auto*
next
case *2*
hence $x \neq y$ $x \notin fvs\ B$ **and** A: $A = [x \leftrightarrow y] \cdot B$ **by** *auto*
hence $x \notin fvs\ B'$ **using** ‹$B >>> B'$› *complete-development-fvs* **by** *auto*
obtain A' **where** A': $A' = [x \leftrightarrow y] \cdot B'$ **by** *auto*
hence $A >>> A'$ **using** A ‹$B >>> B'$› *complete-development-prm* **by** *auto*
have $X = Fn\ x\ T\ A'$
using *fn-eq* ‹$x \neq y$› ‹$x \notin fvs\ B'$› A' ‹$X = Fn\ y\ S\ B'$› ‹$T = S$› **by** *metis*
thus *?thesis* **using** ‹$A >>> A'$› **by** *auto*
next
qed
next
qed (
metis unit-not-fn,
metis var-not-fn,
metis app-not-fn,
metis app-not-fn,
metis fn-not-pair,
metis fn-not-fst,
metis fn-not-fst,
metis fn-not-snd,
metis fn-not-snd
)

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

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

```
  hence (Var x) >>> X using complete-development.var by metis
  thus ?case by auto
next
case (3 A B)
  from this obtain A′ B′ where A′: A >>> A′ and B′: B >>> B′ by auto
  consider (∃x T M. A = Fn x T M) | ¬(∃x T M. A = Fn x T M) by auto
  thus ?case proof(cases)
    case 1
      from this obtain x T M where A: A = Fn x T M by auto
      from this obtain M′ where A′ = Fn x T M′ and M >>> M′
        using complete-development-fnE A′ by metis
      obtain X where X = (M′[x ::= B′]) by auto
      hence (App A B) >>> X
        using complete-development.beta ‹M >>> M′› B′ A by metis
      thus ?thesis by auto
    next
    case 2
      thus ?thesis using complete-development.app A′ B′ by metis
    next
  qed
next
case (4 x T A)
  from this obtain A′ where A′: A >>> A′ by auto
  obtain X where X = Fn x T A′ by auto
  hence (Fn x T A) >>> X using complete-development.eta A′ by metis
  thus ?case by auto
next
case (5 A B)
  thus ?case using complete-development.pair by blast
next
case (6 P)
  consider ∃A B. P = Pair A B | ∄A B. P = Pair A B by auto
  thus ?case proof(cases)
    case 1
     from this obtain A B X where P = Pair A B P >>> X using 6 by auto
      from this obtain A′ B′ where A >>> A′ B >>> B′ X = Pair A′ B′
        using complete-development-pairE by metis
      thus ?thesis using complete-development.fst2 ‹P = Pair A B› by metis
    next
    case 2
      thus ?thesis using complete-development.fst1 6 by blast
    next
  qed
next
case (7 P)
  consider ∃A B. P = Pair A B | ∄A B. P = Pair A B by auto
  thus ?case proof(cases)
    case 1
     from this obtain A B X where P = Pair A B P >>> X using 7 by auto
```

from *this* **obtain** A' B' **where** $A >>> A'$ $B >>> B'$ $X = Pair\ A'\ B'$
using *complete-development-pairE* **by** *metis*
thus *?thesis* **using** *complete-development.snd2* ‹$P = Pair\ A\ B$› **by** *metis*
next
case *2*
thus *?thesis* **using** *complete-development.snd1 7* **by** *blast*
next
qed
next
qed

lemma *complete-development-triangle*:
assumes $A >>> D$ $A >> B$
shows $B >> D$
using *assms* **proof**(*induction arbitrary*: B *rule*: *complete-development.induct*)
case *unit*
thus *?case* **using** *parallel-reduction-unitE parallel-reduction.refl* **by** *metis*
next
case (*var* x)
thus *?case* **using** *parallel-reduction-varE parallel-reduction.refl* **by** *metis*
next
case (*beta* A A' C C' x T)
hence $A >> A'$ $C >> C'$ **using** *parallel-reduction.refl* **by** *metis+*
from ‹(App ($Fn\ x\ T\ A$) C) $>> B$› **consider**
$B = App\ (Fn\ x\ T\ A)\ C$
| $\exists A''\ C''.\ (A >> A'') \wedge (C >> C'') \wedge B = (A''[x ::= C''])$
| $\exists A''\ C''.\ ((Fn\ x\ T\ A) >> (Fn\ x\ T\ A'')) \wedge (C >> C'') \wedge B = (App\ (Fn\ x$
$T\ A'')\ C'')$
using *parallel-reduction-redexE* **by** *metis*
thus *?case* **proof**(*cases*)
case *1*
thus *?thesis* **using** *parallel-reduction.beta* ‹$A >> A'$› ‹$C >> C'$› **by** *metis*
next
case *2*
from *this* **obtain** A'' C'' **where** $A >> A''$ $C >> C''$ **and** B: $B = (A''[x$
$::= C''])$ **by** *auto*
hence $A'' >> A'$ $C'' >> C'$ **using** *beta.IH* **by** *metis+*
thus *?thesis* **using** B *parallel-reduction-subst* **by** *metis*
next
case *3*
from *this* **obtain** A'' C''
where ($Fn\ x\ T\ A$) $>>$ ($Fn\ x\ T\ A''$) $C >> C''$ **and** B: $B = App\ (Fn\ x\ T$
$A'')\ C''$
by *auto*
hence $C'' >> C'$ **using** *beta.IH* **by** *metis*
have $A'' >> A'$
proof −
thm *parallel-reduction-fnE*
from ‹($Fn\ x\ T\ A$) $>>$ ($Fn\ x\ T\ A''$)› **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
    qed
    thus ?thesis using B ‹C'' >> C'› parallel-reduction.beta by metis
  next
 qed
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
 qed
next
case (app A A' C C')
  from this obtain A'' C'' where A >> A'' C >> C'' and B: 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
case (fst1 P P')
 from this obtain P'' where P >> P'' B = Fst P''
```

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

lemma *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 *?thesis* **by** *blast*
qed

inductive *parallel-reduces* :: $'a\ trm \Rightarrow 'a\ trm \Rightarrow bool$ (- $>>^*$ -) **where**
reflexive: $A >>^* A$
| *transitive*: $[\![A >>^* A'; A' >> A'']\!] \Longrightarrow A >>^* A''$

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

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

lemma *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 →β* 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) →β* (App (Fn x T A′) B′)
      using ‹A →β* 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′) →β (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 $(A >>^* B) = (A \rightarrow\beta^* B)$
proof −
have →: $(A >>^* B) \Longrightarrow (A \rightarrow\beta^* B)$
proof(*induction rule*: *parallel-reduces.induct*)
case (*reflexive A*)
thus *?case* **using** *beta-reduces.reflexive*.
next
case (*transitive A A' A''*)
thus *?case* **using** *beta-reduces-composition parallel-reduction-is-beta-reduction* **by** *metis*
next
qed

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

from ← → **show** $(A >>^* B) = (A \rightarrow\beta^* B)$ **by** *blast*
qed

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

end
end