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

Michael Rawson

September 30, 2020

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* ⇒ *'a*
  assumes *finite S* ⟹ *fresh-in S* ∉ *S*

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

  instance ⟨*proof*⟩
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*⟩

lemma *preprm-ext-reflexive*:
  shows *x* =*p x*
⟨*proof*⟩

corollary *preprm-ext-reflp*:
  shows *reflp preprm-ext*
⟨*proof*⟩

lemma *preprm-ext-symmetric*:
  assumes *x* =*p y*
  shows *y* =*p x*
⟨*proof*⟩

corollary *preprm-ext-symp*:
  shows *symp preprm-ext*
⟨*proof*⟩

lemma *preprm-ext-transitive*:
  assumes *x* =*p y* **and** *y* =*p z*
  shows *x* =*p z*
⟨*proof*⟩

corollary *preprm-ext-transp*:

shows *transp preprm-ext*
⟨*proof*⟩

lemma *preprm-apply-composition*:
shows *preprm-apply* (*preprm-compose f g*) *x* = *preprm-apply f* (*preprm-apply g x*)
⟨*proof*⟩

lemma *preprm-apply-unequal*:
assumes $x \neq y$
shows *preprm-apply* π *x* $\neq$ *preprm-apply* π *y*
⟨*proof*⟩

lemma *preprm-unit-equal-id*:
shows *preprm-unit a a* =*p preprm-id*
⟨*proof*⟩

lemma *preprm-unit-inaction*:
assumes $x \neq a$ **and** $x \neq b$
shows *preprm-apply* (*preprm-unit a b*) *x* = *x*
⟨*proof*⟩

lemma *preprm-unit-action*:
shows *preprm-apply* (*preprm-unit a b*) *a* = *b*
⟨*proof*⟩

lemma *preprm-unit-commutes*:
shows *preprm-unit a b* =*p preprm-unit b a*
⟨*proof*⟩

lemma *preprm-singleton-involution*:
shows *preprm-compose* [*s*] [*s*] =*p preprm-id*
⟨*proof*⟩

lemma *preprm-unit-involution*:
shows *preprm-compose* (*preprm-unit a b*) (*preprm-unit a b*) =*p preprm-id*
⟨*proof*⟩

lemma *preprm-apply-id*:
shows *preprm-apply preprm-id x* = *x*
⟨*proof*⟩

lemma *preprm-apply-injective*:
shows *inj* (*preprm-apply* π)
⟨*proof*⟩

lemma *preprm-disagreement-composition*:
assumes $a \neq b$ $b \neq c$ $a \neq c$
shows {*x*.

preprm-apply (*preprm-compose* (*preprm-unit a b*) (*preprm-unit b c*)) *x* ≠
preprm-apply (*preprm-unit a c*) *x*
} = {*a*, *b*}
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *preprm-ext-compose-left*:
assumes *P* =*p S*
shows *preprm-compose π P* =*p preprm-compose π S*
⟨*proof*⟩

lemma *preprm-ext-compose-right*:
assumes *P* =*p S*
shows *preprm-compose P π* =*p preprm-compose S π*
⟨*proof*⟩

lemma *preprm-ext-uncompose*:
assumes *π* =*p σ preprm-compose π P* =*p preprm-compose σ S*
shows *P* =*p S*
⟨*proof*⟩

lemma *preprm-inv-compose*:
shows *preprm-compose* (*preprm-inv π*) *π* =*p preprm-id*
⟨*proof*⟩

lemma *preprm-inv-involution*:
shows *preprm-inv* (*preprm-inv π*) = *π*
⟨*proof*⟩

lemma *preprm-inv-ext*:
assumes *π* =*p σ*
shows *preprm-inv π* =*p preprm-inv σ*
⟨*proof*⟩

quotient-type *'a prm* = *'a preprm* / *preprm-ext*
⟨*proof*⟩

lift-definition *prm-id* :: *'a prm* (*ε*) **is** *preprm-id*⟨*proof*⟩

lift-definition *prm-apply* :: *'a prm* ⇒ *'a* ⇒ *'a* (**infix** $ *140*) **is** *preprm-apply*
⟨*proof*⟩

lift-definition *prm-compose* :: *'a prm* ⇒ *'a prm* ⇒ *'a prm* (**infixr** ⋄ *145*) **is**

preprm-compose
⟨*proof*⟩

lift-definition *prm-unit* :: *'a* ⇒ *'a* ⇒ *'a prm* ([- ↔ -]) **is** *preprm-unit*⟨*proof*⟩

lift-definition *prm-inv* :: *'a prm* ⇒ *'a prm* **is** *preprm-inv*
⟨*proof*⟩

lemma *prm-apply-composition*:
fixes *f g* :: *'a prm* **and** *x* :: *'a*
shows *f* ⋄ *g* $ *x* = *f* $ (*g* $ *x*)
⟨*proof*⟩

lemma *prm-apply-unequal*:
fixes *π* :: *'a prm* **and** *x y* :: *'a*
assumes *x* ≠ *y*
shows *π* $ *x* ≠ *π* $ *y*
⟨*proof*⟩

lemma *prm-unit-equal-id*:
fixes *a* :: *'a*
shows [*a* ↔ *a*] = *ε*
⟨*proof*⟩

lemma *prm-unit-inaction*:
fixes *a b x* :: *'a*
assumes *x* ≠ *a* **and** *x* ≠ *b*
shows [*a* ↔ *b*] $ *x* = *x*
⟨*proof*⟩

lemma *prm-unit-action*:
fixes *a b* :: *'a*
shows [*a* ↔ *b*] $ *a* = *b*
⟨*proof*⟩

lemma *prm-unit-commutes*:
fixes *a b* :: *'a*
shows [*a* ↔ *b*] = [*b* ↔ *a*]
⟨*proof*⟩

lemma *prm-unit-involution*:
fixes *a b* :: *'a*
shows [*a* ↔ *b*] ⋄ [*a* ↔ *b*] = *ε*
⟨*proof*⟩

lemma *prm-apply-id*:
fixes *x* :: *'a*
shows *ε* $ *x* = *x*
⟨*proof*⟩

lemma *prm-apply-injective*:
shows *inj* (*prm-apply* π)
⟨*proof*⟩

lemma *prm-inv-compose*:
shows (*prm-inv* π) $\diamond$ π = ε
⟨*proof*⟩

interpretation *'a prm*: *semigroup prm-compose*
⟨*proof*⟩

interpretation *'a prm*: *group prm-compose prm-id prm-inv*
⟨*proof*⟩

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

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

lemma *prm-set-membership*:
assumes $x \in S$
shows π $ $x \in \pi$ {$} S
⟨*proof*⟩

lemma *prm-set-notmembership*:
assumes $x \notin S$
shows π $ $x \notin \pi$ {$} S
⟨*proof*⟩

lemma *prm-set-singleton*:
shows π {$} {$x$} = {$\pi$ $ x}
⟨*proof*⟩

lemma *prm-set-id*:
shows ε {$} S = S
⟨*proof*⟩

lemma *prm-set-unit-inaction*:
assumes $a \notin S$ **and** $b \notin S$
shows [$a \leftrightarrow b$] {$} S = S
⟨*proof*⟩

lemma *prm-set-unit-action*:
assumes $a \in S$ **and** $b \notin S$
shows [$a \leftrightarrow b$] {$} S = $S - \{a\} \cup \{b\}$
⟨*proof*⟩

lemma *prm-set-distributes-union*:
shows $\pi\ \{\$\}\ (S \cup T) = (\pi\ \{\$\}\ S) \cup (\pi\ \{\$\}\ T)$
⟨*proof*⟩

lemma *prm-set-distributes-difference*:
shows $\pi\ \{\$\}\ (S - T) = (\pi\ \{\$\}\ S) - (\pi\ \{\$\}\ T)$
⟨*proof*⟩

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 $x \in ds\ \pi\ \sigma \equiv \pi\ \$\ x \neq \sigma\ \$\ x$
⟨*proof*⟩

lemma *prm-disagreement-composition*:
assumes $a \neq b\ b \neq c\ a \neq c$
shows $ds\ ([a \leftrightarrow b] \diamond [b \leftrightarrow c])\ [a \leftrightarrow c] = \{a, b\}$
⟨*proof*⟩

lemma *prm-compose-push*:
shows $\pi \diamond [a \leftrightarrow b] = [\pi\ \$\ a \leftrightarrow \pi\ \$\ b] \diamond \pi$
⟨*proof*⟩

end
theory *PreSimplyTyped*
imports *Fresh Permutation*
begin

type-synonym *tvar* = *nat*

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

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

fun *ptrm-fvs* :: $'a\ ptrm \Rightarrow 'a\ set$ **where**
ptrm-fvs PUnit = {}

| *ptrm-fvs* (*PVar x*) = {*x*}
| *ptrm-fvs* (*PApp A B*) = *ptrm-fvs A* ∪ *ptrm-fvs B*
| *ptrm-fvs* (*PFn x - A*) = *ptrm-fvs A* − {*x*}
| *ptrm-fvs* (*PPair A B*) = *ptrm-fvs A* ∪ *ptrm-fvs B*
| *ptrm-fvs* (*PFst P*) = *ptrm-fvs P*
| *ptrm-fvs* (*PSnd P*) = *ptrm-fvs P*

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

inductive *ptrm-alpha-equiv* :: *'a ptrm* ⇒ *'a ptrm* ⇒ *bool* (**infix** ≈ *100*) **where**
  unit: *PUnit* ≈ *PUnit*
| *var*: (*PVar x*) ≈ (*PVar x*)
| *app*: ⟦*A* ≈ *B*; *C* ≈ *D*⟧ ⟹ (*PApp A C*) ≈ (*PApp B D*)
| *fn1*: *A* ≈ *B* ⟹ (*PFn x T A*) ≈ (*PFn x T B*)
| *fn2*: ⟦*a* ≠ *b*; *A* ≈ [*a* ↔ *b*] · *B*; *a* ∉ *ptrm-fvs B*⟧ ⟹ (*PFn a T A*) ≈ (*PFn b T B*)
| *pair*: ⟦*A* ≈ *B*; *C* ≈ *D*⟧ ⟹ (*PPair A C*) ≈ (*PPair B D*)
| *fst*: *A* ≈ *B* ⟹ *PFst A* ≈ *PFst B*
| *snd*: *A* ≈ *B* ⟹ *PSnd A* ≈ *PSnd B*

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

lemma *ptrm-prm-apply-id*:
  shows *ε* · *X* = *X*
⟨*proof*⟩

lemma *ptrm-prm-apply-compose*:
  shows *π* · *σ* · *X* = (*π* ⋄ *σ*) · *X*
⟨*proof*⟩

lemma *ptrm-size-prm*:
  shows *size X* = *size* (*π* · *X*)
⟨*proof*⟩

lemma *ptrm-size-alpha-equiv*:
assumes $X \approx Y$
shows *size* X = *size* Y
⟨*proof*⟩

lemma *ptrm-fvs-finite*:
shows *finite* (*ptrm-fvs* X)
⟨*proof*⟩

lemma *ptrm-prm-fvs*:
shows *ptrm-fvs* (π · X) = π {$} *ptrm-fvs* X
⟨*proof*⟩

lemma *ptrm-alpha-equiv-fvs*:
assumes $X \approx Y$
shows *ptrm-fvs* X = *ptrm-fvs* Y
⟨*proof*⟩

lemma *ptrm-alpha-equiv-prm*:
assumes $X \approx Y$
shows π · $X \approx \pi$ · Y
⟨*proof*⟩

lemma *ptrm-swp-transfer*:
shows $[a \leftrightarrow b]$ · $X \approx Y \longleftrightarrow X \approx [a \leftrightarrow b]$ · Y
⟨*proof*⟩

lemma *ptrm-alpha-equiv-fvs-transfer*:
assumes $A \approx [a \leftrightarrow b]$ · B **and** $a \notin$ *ptrm-fvs* B
shows $b \notin$ *ptrm-fvs* A
⟨*proof*⟩

lemma *ptrm-prm-agreement-equiv*:
assumes $\bigwedge a.\ a \in ds\ \pi\ \sigma \Longrightarrow a \notin$ *ptrm-fvs* M
shows π · $M \approx \sigma$ · M
⟨*proof*⟩

lemma *ptrm-prm-unit-inaction*:
assumes $a \notin$ *ptrm-fvs* X $b \notin$ *ptrm-fvs* X
shows $[a \leftrightarrow b]$ · $X \approx X$
⟨*proof*⟩

lemma *ptrm-alpha-equiv-reflexive*:
shows $M \approx M$
⟨*proof*⟩

corollary *ptrm-alpha-equiv-reflp*:
shows *reflp ptrm-alpha-equiv*
⟨*proof*⟩

lemma *ptrm-alpha-equiv-symmetric*:
  assumes $X \approx Y$
  shows $Y \approx X$
⟨*proof*⟩

corollary *ptrm-alpha-equiv-symp*:
  shows *symp ptrm-alpha-equiv*
⟨*proof*⟩

lemma *ptrm-alpha-equiv-transitive*:
  assumes $X \approx Y$ **and** $Y \approx Z$
  shows $X \approx Z$
⟨*proof*⟩

corollary *ptrm-alpha-equiv-transp*:
  shows *transp ptrm-alpha-equiv*
⟨*proof*⟩

type-synonym *'a typing-ctx* = *'a* ⇀ *type*

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

lemma *ptrm-infer-type-swp-types*:

assumes $a \neq b$
shows *ptrm-infer-type* $(\Gamma(a \mapsto T,\ b \mapsto S))\ X$ = *ptrm-infer-type* $(\Gamma(a \mapsto S,\ b \mapsto T))\ ([a \leftrightarrow b] \cdot X)$
⟨*proof*⟩

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* $(\Gamma(b \mapsto \tau))\ ([a \leftrightarrow b] \cdot X)$
⟨*proof*⟩

lemma *ptrm-infer-type-alpha-equiv*:
assumes $X \approx Y$
shows *ptrm-infer-type* $\Gamma\ X$ = *ptrm-infer-type* $\Gamma\ Y$
⟨*proof*⟩

end
theory *SimplyTyped*
imports *PreSimplyTyped*
begin

quotient-type *'a trm* = *'a ptrm* / *ptrm-alpha-equiv*
⟨*proof*⟩

lift-definition *Unit* :: *'a trm* **is** *PUnit*⟨*proof*⟩
lift-definition *Var* :: *'a* ⇒ *'a trm* **is** *PVar*⟨*proof*⟩
lift-definition *App* :: *'a trm* ⇒ *'a trm* ⇒ *'a trm* **is** *PApp* ⟨*proof*⟩
lift-definition *Fn* :: *'a* ⇒ *type* ⇒ *'a trm* ⇒ *'a trm* **is** *PFn* ⟨*proof*⟩
lift-definition *Pair* :: *'a trm* ⇒ *'a trm* ⇒ *'a trm* **is** *PPair* ⟨*proof*⟩
lift-definition *Fst* :: *'a trm* ⇒ *'a trm* **is** *PFst* ⟨*proof*⟩
lift-definition *Snd* :: *'a trm* ⇒ *'a trm* **is** *PSnd* ⟨*proof*⟩
lift-definition *fvs* :: *'a trm* ⇒ *'a set* **is** *ptrm-fvs* ⟨*proof*⟩
lift-definition *prm* :: *'a prm* ⇒ *'a trm* ⇒ *'a trm* (**infixr** · *150*) **is** *ptrm-apply-prm* ⟨*proof*⟩
lift-definition *depth* :: *'a trm* ⇒ *nat* **is** *size* ⟨*proof*⟩

lemma *depth-prm*:
shows *depth* $(\pi \cdot A)$ = *depth* A
⟨*proof*⟩

lemma *depth-app*:
shows *depth* $A <$ *depth* $(App\ A\ B)$ *depth* $B <$ *depth* $(App\ A\ B)$
⟨*proof*⟩

lemma *depth-fn*:
shows *depth* $A <$ *depth* $(Fn\ x\ T\ A)$
⟨*proof*⟩

lemma *depth-pair*:

shows *depth A < depth (Pair A B) depth B < depth (Pair A B)*
⟨*proof*⟩

lemma *depth-fst*:
shows *depth P < depth (Fst P)*
⟨*proof*⟩

lemma *depth-snd*:
shows *depth P < depth (Snd P)*
⟨*proof*⟩

lemma *unit-not-var*:
shows *Unit ≠ Var x*
⟨*proof*⟩

lemma *unit-not-app*:
shows *Unit ≠ App A B*
⟨*proof*⟩

lemma *unit-not-fn*:
shows *Unit ≠ Fn x T A*
⟨*proof*⟩

lemma *unit-not-pair*:
shows *Unit ≠ Pair A B*
⟨*proof*⟩

lemma *unit-not-fst*:
shows *Unit ≠ Fst P*
⟨*proof*⟩

lemma *unit-not-snd*:
shows *Unit ≠ Snd P*
⟨*proof*⟩

lemma *var-not-app*:
shows *Var x ≠ App A B*
⟨*proof*⟩

lemma *var-not-fn*:
shows *Var x ≠ Fn y T A*
⟨*proof*⟩

lemma *var-not-pair*:
shows *Var x ≠ Pair A B*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *var-not-snd*:
  shows *Var x* ≠ *Snd P*
⟨*proof*⟩

lemma *app-not-fn*:
  shows *App A B* ≠ *Fn y T X*
⟨*proof*⟩

lemma *app-not-pair*:
  shows *App A B* ≠ *Pair C D*
⟨*proof*⟩

lemma *app-not-fst*:
  shows *App A B* ≠ *Fst P*
⟨*proof*⟩

lemma *app-not-snd*:
  shows *App A B* ≠ *Snd P*
⟨*proof*⟩

lemma *fn-not-pair*:
  shows *Fn x T A* ≠ *Pair C D*
⟨*proof*⟩

lemma *fn-not-fst*:
  shows *Fn x T A* ≠ *Fst P*
⟨*proof*⟩

lemma *fn-not-snd*:
  shows *Fn x T A* ≠ *Snd P*
⟨*proof*⟩

lemma *pair-not-fst*:
  shows *Pair A B* ≠ *Fst P*
⟨*proof*⟩

lemma *pair-not-snd*:
  shows *Pair A B* ≠ *Snd P*
⟨*proof*⟩

lemma *fst-not-snd*:
  shows *Fst P* ≠ *Snd Q*
⟨*proof*⟩

lemma *trm-simp*:
  shows
    Var x = *Var y* ⟹ *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*⟩

lemma *fn-eq*:
assumes $x \neq y\ x \notin fvs\ B\ A = [x \leftrightarrow y] \cdot B$
shows $Fn\ x\ T\ A = Fn\ y\ T\ B$
⟨*proof*⟩

lemma *trm-prm-simp*:
shows
$\pi \cdot Unit = Unit$
$\pi \cdot Var\ x = Var\ (\pi\ \$\ x)$
$\pi \cdot App\ A\ B = App\ (\pi \cdot A)\ (\pi \cdot B)$
$\pi \cdot Fn\ x\ T\ A = Fn\ (\pi\ \$\ x)\ T\ (\pi \cdot A)$
$\pi \cdot Pair\ A\ B = Pair\ (\pi \cdot A)\ (\pi \cdot B)$
$\pi \cdot Fst\ P = Fst\ (\pi \cdot P)$
$\pi \cdot Snd\ P = Snd\ (\pi \cdot P)$
⟨*proof*⟩

lemma *trm-prm-apply-compose*:
shows $\pi \cdot \sigma \cdot A = (\pi \diamond \sigma) \cdot A$
⟨*proof*⟩

lemma *fvs-finite*:
shows *finite* $(fvs\ M)$
⟨*proof*⟩

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

lemma *var-prm-action*:
shows $[a \leftrightarrow b] \cdot Var\ a = Var\ b$
⟨*proof*⟩

lemma *var-prm-inaction*:
assumes $a \neq x$ $b \neq x$
shows $[a \leftrightarrow b] \cdot \mathit{Var}\ x = \mathit{Var}\ x$
⟨*proof*⟩

lemma *trm-prm-apply-id*:
shows $\varepsilon \cdot M = M$
⟨*proof*⟩

lemma *trm-prm-unit-inaction*:
assumes $a \notin \mathit{fvs}\ X$ $b \notin \mathit{fvs}\ X$
shows $[a \leftrightarrow b] \cdot X = X$
⟨*proof*⟩

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

lemma *trm-induct*:
fixes $P :: 'a\ \mathit{trm} \Rightarrow \mathit{bool}$
assumes
$P\ \mathit{Unit}$
$\bigwedge x.\ P\ (\mathit{Var}\ x)$
$\bigwedge A\ B.\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (\mathit{App}\ A\ B)$
$\bigwedge x\ T\ A.\ P\ A \Longrightarrow P\ (\mathit{Fn}\ x\ T\ A)$
$\bigwedge A\ B.\ [\![P\ A;\ P\ B]\!] \Longrightarrow P\ (\mathit{Pair}\ A\ B)$
$\bigwedge A.\ P\ A \Longrightarrow P\ (\mathit{Fst}\ A)$
$\bigwedge A.\ P\ A \Longrightarrow P\ (\mathit{Snd}\ A)$
shows $P\ M$
⟨*proof*⟩

lemma *trm-cases*:
assumes
$M = \mathit{Unit} \Longrightarrow P\ M$
$\bigwedge x.\ M = \mathit{Var}\ x \Longrightarrow P\ M$
$\bigwedge A\ B.\ M = \mathit{App}\ A\ B \Longrightarrow P\ M$
$\bigwedge x\ T\ A.\ M = \mathit{Fn}\ x\ T\ A \Longrightarrow P\ M$
$\bigwedge A\ B.\ M = \mathit{Pair}\ A\ B \Longrightarrow P\ M$
$\bigwedge A.\ M = \mathit{Fst}\ A \Longrightarrow P\ M$
$\bigwedge A.\ M = \mathit{Snd}\ A \Longrightarrow P\ M$
shows $P\ M$
⟨*proof*⟩

lemma *trm-depth-induct*:
assumes
$P\ \mathit{Unit}$
$\bigwedge x.\ P\ (\mathit{Var}\ x)$

$\bigwedge A\ B.\ [\![\bigwedge K.\ depth\ K < depth\ (App\ A\ B) \Longrightarrow P\ K]\!] \Longrightarrow P\ (App\ A\ B)$
$\bigwedge M\ x\ T\ A.\ (\bigwedge K.\ depth\ K < depth\ (Fn\ x\ T\ A) \Longrightarrow P\ K) \Longrightarrow P\ (Fn\ x\ T\ A)$
$\bigwedge A\ B.\ [\![\bigwedge K.\ depth\ K < depth\ (Pair\ A\ B) \Longrightarrow P\ K]\!] \Longrightarrow P\ (Pair\ A\ B)$
$\bigwedge A.\ [\![\bigwedge K.\ depth\ K < depth\ (Fst\ A) \Longrightarrow P\ K]\!] \Longrightarrow P\ (Fst\ A)$
$\bigwedge A.\ [\![\bigwedge K.\ depth\ K < depth\ (Snd\ A) \Longrightarrow P\ K]\!] \Longrightarrow P\ (Snd\ A)$
shows $P\ M$
⟨*proof*⟩

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*⟩

lemma *trm-strong-induct*:
fixes $P :: 'a\ set \Rightarrow 'a\ trm \Rightarrow bool$
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*⟩

lemma *trm-strong-cases*:
fixes $P :: 'a\ set \Rightarrow 'a\ trm \Rightarrow bool$
assumes
$M = Unit \Longrightarrow P\ S\ M$
$\bigwedge x.\ M = Var\ x \Longrightarrow P\ S\ M$
$\bigwedge A\ B.\ M = App\ A\ B \Longrightarrow P\ S\ M$
$\bigwedge x\ T\ A.\ M = Fn\ x\ T\ A \Longrightarrow x \notin S \Longrightarrow P\ S\ M$
$\bigwedge A\ B.\ M = Pair\ A\ B \Longrightarrow P\ S\ M$
$\bigwedge A.\ M = Fst\ A \Longrightarrow P\ S\ M$
$\bigwedge A.\ M = Snd\ A \Longrightarrow P\ S\ M$
finite S
shows $P\ S\ M$
⟨*proof*⟩

lemma *trm-strong-depth-induct*:
fixes $P :: 'a\ set \Rightarrow 'a\ trm \Rightarrow bool$
assumes
$P\ S\ Unit$
$\bigwedge x.\ P\ S\ (Var\ x)$

$\bigwedge A\ B.\ [\![\bigwedge K.\ depth\ K < depth\ (App\ A\ B) \Longrightarrow P\ S\ K]\!] \Longrightarrow P\ S\ (App\ A\ B)$
$\bigwedge x\ T.\ x \notin S \Longrightarrow (\bigwedge A.\ (\bigwedge K.\ depth\ K < depth\ (Fn\ x\ T\ A) \Longrightarrow P\ S\ K) \Longrightarrow P\ S\ (Fn\ x\ T\ A))$
$\bigwedge A\ B.\ [\![\bigwedge K.\ depth\ K < depth\ (Pair\ A\ B) \Longrightarrow P\ S\ K]\!] \Longrightarrow P\ S\ (Pair\ A\ B)$
$\bigwedge A.\ [\![\bigwedge K.\ depth\ K < depth\ (Fst\ A) \Longrightarrow P\ S\ K]\!] \Longrightarrow P\ S\ (Fst\ A)$
$\bigwedge A.\ [\![\bigwedge K.\ depth\ K < depth\ (Snd\ A) \Longrightarrow P\ S\ K]\!] \Longrightarrow P\ S\ (Snd\ A)$
finite S
shows *P S M*
⟨*proof*⟩

lemma *trm-prm-fvs*:
shows *fvs* $(\pi \cdot M) = \pi\ \{\$\}\ fvs\ M$
⟨*proof*⟩

inductive *typing* :: $'a$ *typing-ctx* ⇒ $'a$ *trm* ⇒ *type* ⇒ *bool* (- ⊢ - : -) **where**
tunit: $\Gamma \vdash Unit : TUnit$
| *tvar*: $\Gamma\ x = Some\ \tau \Longrightarrow \Gamma \vdash Var\ x : \tau$
| *tapp*: $[\![\Gamma \vdash f : (TArr\ \tau\ \sigma); \Gamma \vdash x : \tau]\!] \Longrightarrow \Gamma \vdash App\ f\ x : \sigma$
| *tfn*: $\Gamma(x \mapsto \tau) \vdash A : \sigma \Longrightarrow \Gamma \vdash Fn\ x\ \tau\ A : (TArr\ \tau\ \sigma)$
| *tpair*: $[\![\Gamma \vdash A : \tau; \Gamma \vdash B : \sigma]\!] \Longrightarrow \Gamma \vdash Pair\ A\ B : (TPair\ \tau\ \sigma)$
| *tfst*: $\Gamma \vdash P : (TPair\ \tau\ \sigma) \Longrightarrow \Gamma \vdash Fst\ P : \tau$
| *tsnd*: $\Gamma \vdash P : (TPair\ \tau\ \sigma) \Longrightarrow \Gamma \vdash Snd\ P : \sigma$

lemma *typing-prm*:
assumes $\Gamma \vdash M : \tau\ \bigwedge y.\ y \in fvs\ M \Longrightarrow \Gamma\ y = \Delta\ (\pi\ \$\ y)$
shows $\Delta \vdash \pi \cdot M : \tau$
⟨*proof*⟩

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

lemma *typing-unitE*:
assumes $\Gamma \vdash Unit : \tau$
shows $\tau = TUnit$
⟨*proof*⟩

lemma *typing-varE*:
assumes $\Gamma \vdash Var\ x : \tau$
shows $\Gamma\ x = Some\ \tau$
⟨*proof*⟩

lemma *typing-appE*:
assumes $\Gamma \vdash App\ A\ B : \sigma$
shows $\exists \tau.\ (\Gamma \vdash A : (TArr\ \tau\ \sigma)) \wedge (\Gamma \vdash B : \tau)$
⟨*proof*⟩

lemma *typing-fnE*:

assumes $\Gamma \vdash \mathit{Fn}\ x\ T\ A : \vartheta$
shows $\exists \sigma.\ \vartheta = (\mathit{TArr}\ T\ \sigma) \wedge (\Gamma(x \mapsto T) \vdash A : \sigma)$
⟨*proof*⟩

lemma *typing-pairE*:
assumes $\Gamma \vdash \mathit{Pair}\ A\ B : \vartheta$
shows $\exists \tau\ \sigma.\ \vartheta = (\mathit{TPair}\ \tau\ \sigma) \wedge (\Gamma \vdash A : \tau) \wedge (\Gamma \vdash B : \sigma)$
⟨*proof*⟩

lemma *typing-fstE*:
assumes $\Gamma \vdash \mathit{Fst}\ P : \tau$
shows $\exists \sigma.\ (\Gamma \vdash P : (\mathit{TPair}\ \tau\ \sigma))$
⟨*proof*⟩

lemma *typing-sndE*:
assumes $\Gamma \vdash \mathit{Snd}\ P : \sigma$
shows $\exists \tau.\ (\Gamma \vdash P : (\mathit{TPair}\ \tau\ \sigma))$
⟨*proof*⟩

theorem *typing-weaken*:
assumes $\Gamma \vdash M : \tau\ y \notin \mathit{fvs}\ M$
shows $\Gamma(y \mapsto \sigma) \vdash M : \tau$
⟨*proof*⟩

lift-definition *infer* :: *'a typing-ctx* ⇒ *'a trm* ⇒ *type option* **is** *ptrm-infer-type*
⟨*proof*⟩

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
  )
⟨proof⟩

lemma infer-unitE:
  assumes infer Γ Unit = Some τ
  shows τ = TUnit
⟨proof⟩

lemma infer-varE:
  assumes infer Γ (Var x) = Some τ
  shows Γ x = Some τ
⟨proof⟩

lemma infer-appE:
  assumes infer Γ (App A B) = Some τ
  shows ∃σ. infer Γ A = Some (TArr σ τ) ∧ infer Γ B = Some σ
⟨proof⟩

lemma infer-fnE:
  assumes infer Γ (Fn x T A) = Some τ
  shows ∃σ. τ = TArr T σ ∧ infer (Γ(x ↦ T)) A = Some σ
⟨proof⟩

lemma infer-pairE:
  assumes infer Γ (Pair A B) = Some τ
  shows ∃ T S. τ = TPair T S ∧ infer Γ A = Some T ∧ infer Γ B = Some S
⟨proof⟩

lemma infer-fstE:
  assumes infer Γ (Fst P) = Some τ
  shows ∃ T S. infer Γ P = Some (TPair T S) ∧ τ = T
⟨proof⟩

lemma infer-sndE:
  assumes infer Γ (Snd P) = Some τ
  shows ∃ T S. infer Γ P = Some (TPair T S) ∧ τ = S
⟨proof⟩

lemma infer-sound:
  assumes infer Γ M = Some τ
  shows Γ ⊢ M : τ
⟨proof⟩

lemma infer-complete:
  assumes Γ ⊢ M : τ
```

shows *infer* Γ *M* = *Some* τ
⟨*proof*⟩

theorem *infer-valid*:
shows (Γ ⊢ *M* : τ) = (*infer* Γ *M* = *Some* τ)
⟨*proof*⟩

inductive *substitutes* :: 'a *trm* ⇒ 'a ⇒ 'a *trm* ⇒ 'a *trm* ⇒ *bool* **where**
unit: *substitutes Unit y M Unit*
| *var1*: *x* = *y* ⟹ *substitutes* (*Var x*) *y M M*
| *var2*: *x* ≠ *y* ⟹ *substitutes* (*Var x*) *y M* (*Var x*)
| *app*: ⟦*substitutes A x M A'*; *substitutes B x M B'*⟧ ⟹ *substitutes* (*App A B*) *x M* (*App A' B'*)
| *fn*: ⟦*x* ≠ *y*; *y* ∉ *fvs M*; *substitutes A x M A'*⟧ ⟹ *substitutes* (*Fn y T A*) *x M* (*Fn y T A'*)
| *pair*: ⟦*substitutes A x M A'*; *substitutes B x M B'*⟧ ⟹ *substitutes* (*Pair A B*) *x M* (*Pair A' B'*)
| *fst*: *substitutes P x M P'* ⟹ *substitutes* (*Fst P*) *x M* (*Fst P'*)
| *snd*: *substitutes P x M P'* ⟹ *substitutes* (*Snd P*) *x M* (*Snd P'*)

lemma *substitutes-prm*:
assumes *substitutes A x M A'*
shows *substitutes* (π · *A*) (π $ *x*) (π · *M*) (π · *A'*)
⟨*proof*⟩

lemma *substitutes-fvs*:
assumes *substitutes A x M A'*
shows *fvs A'* ⊆ *fvs A* − {*x*} ∪ *fvs M*
⟨*proof*⟩

inductive-cases *substitutes-unitE'*: *substitutes Unit y M X*
lemma *substitutes-unitE*:
assumes *substitutes Unit y M X*
shows *X* = *Unit*
⟨*proof*⟩

inductive-cases *substitutes-varE'*: *substitutes* (*Var x*) *y M X*
lemma *substitutes-varE*:
assumes *substitutes* (*Var x*) *y M X*
shows (*x* = *y* ∧ *M* = *X*) ∨ (*x* ≠ *y* ∧ *X* = *Var x*)
⟨*proof*⟩

inductive-cases *substitutes-appE'*: *substitutes* (*App A B*) *x M X*
lemma *substitutes-appE*:
assumes *substitutes* (*App A B*) *x M X*
shows ∃ *A' B'*. *substitutes A x M A'* ∧ *substitutes B x M B'* ∧ *X* = *App A' B'*
⟨*proof*⟩

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'*
⟨*proof*⟩

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*⟩

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*⟩

inductive-cases *substitutes-sndE'*: *substitutes* (*Snd P*) *x M X*
lemma *substitutes-sndE*:
assumes *substitutes* (*Snd P*) *x M X*
shows $\exists P'$. *substitutes P x M P'* $\wedge$ $X =$ *Snd P'*
⟨*proof*⟩

lemma *substitutes-total*:
shows $\exists X$. *substitutes A x M X*
⟨*proof*⟩

lemma *substitutes-unique*:
assumes *substitutes A x M B substitutes A x M C*
shows $B = C$
⟨*proof*⟩

lemma *substitutes-function*:
shows $\exists!\ X$. *substitutes A x M X*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *subst-simp-var1*:
shows (*Var x*)[*x* ::= *M*] = *M*
⟨*proof*⟩

lemma *subst-simp-var2*:
assumes $x \neq y$

shows $(Var\ x)[y ::= M] = Var\ x$
⟨*proof*⟩

lemma *subst-simp-app*:
shows $(App\ A\ B)[x ::= M] = App\ (A[x ::= M])\ (B[x ::= M])$
⟨*proof*⟩

lemma *subst-simp-fn*:
assumes $x \neq y\ y \notin fvs\ M$
shows $(Fn\ y\ T\ A)[x ::= M] = Fn\ y\ T\ (A[x ::= M])$
⟨*proof*⟩

lemma *subst-simp-pair*:
shows $(Pair\ A\ B)[x ::= M] = Pair\ (A[x ::= M])\ (B[x ::= M])$
⟨*proof*⟩

lemma *subst-simp-fst*:
shows $(Fst\ P)[x ::= M] = Fst\ (P[x ::= M])$
⟨*proof*⟩

lemma *subst-simp-snd*:
shows $(Snd\ P)[x ::= M] = Snd\ (P[x ::= M])$
⟨*proof*⟩

lemma *subst-prm*:
shows $(\pi \cdot (M[z ::= N])) = ((\pi \cdot M)[\pi\ \$\ z ::= \pi \cdot N])$
⟨*proof*⟩

lemma *subst-fvs*:
shows $fvs\ (M[z ::= N]) \subseteq (fvs\ M - \{z\}) \cup fvs\ N$
⟨*proof*⟩

lemma *subst-free*:
assumes $y \notin fvs\ M$
shows $M[y ::= N] = M$
⟨*proof*⟩

lemma *subst-swp*:
assumes $y \notin fvs\ A$
shows $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M])$
⟨*proof*⟩

lemma *barendregt*:
assumes $y \neq z\ y \notin fvs\ L$
shows $M[y ::= N][z ::= L] = (M[z ::= L][y ::= N[z ::= L]])$
⟨*proof*⟩

lemma *typing-subst*:
assumes $\Gamma(z \mapsto \tau) \vdash M : \sigma\ \Gamma \vdash N : \tau$

shows $\Gamma \vdash M[z ::= N] : \sigma$
⟨*proof*⟩

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

lemma *beta-reduction-fvs*:
  assumes $M \to\beta\ M'$
  shows $fvs\ M' \subseteq fvs\ M$
⟨*proof*⟩

lemma *beta-reduction-prm*:
  assumes $M \to\beta\ M'$
  shows $(\pi \cdot M) \to\beta\ (\pi \cdot M')$
⟨*proof*⟩

lemma *beta-reduction-left-unitE*:
  assumes $Unit \to\beta\ M'$
  shows *False*
⟨*proof*⟩

lemma *beta-reduction-left-varE*:
  assumes $(Var\ x) \to\beta\ M'$
  shows *False*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *beta-reduction-left-fnE*:
  assumes $(Fn\ x\ T\ A) \to\beta\ M'$
  shows $\exists A'.\ (A \to\beta\ A') \wedge M' = (Fn\ x\ T\ A')$
⟨*proof*⟩

lemma *beta-reduction-left-pairE*:
assumes $(Pair\ A\ B) \rightarrow\beta\ M'$
shows $(\exists A'.\ (A \rightarrow\beta\ A') \wedge M' = (Pair\ A'\ B)) \vee (\exists B'.\ (B \rightarrow\beta\ B') \wedge M' = (Pair\ A\ B'))$
⟨*proof*⟩

lemma *beta-reduction-left-fstE*:
assumes $(Fst\ P) \rightarrow\beta\ M'$
shows $(\exists P'.\ (P \rightarrow\beta\ P') \wedge M' = (Fst\ P')) \vee (\exists A\ B.\ P = (Pair\ A\ B) \wedge M' = A)$
⟨*proof*⟩

lemma *beta-reduction-left-sndE*:
assumes $(Snd\ P) \rightarrow\beta\ M'$
shows $(\exists P'.\ (P \rightarrow\beta\ P') \wedge M' = (Snd\ P')) \vee (\exists A\ B.\ P = Pair\ A\ B \wedge M' = B)$
⟨*proof*⟩

lemma *preservation'*:
assumes $\Gamma \vdash M : \tau$ **and** $M \rightarrow\beta\ M'$
shows $\Gamma \vdash M' : \tau$
⟨*proof*⟩

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

theorem *progress*:
assumes $\Gamma \vdash M : \tau$
shows $NF\ M \vee (\exists M'.\ (M \rightarrow\beta\ M'))$
⟨*proof*⟩

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

lemma *beta-reduces-composition*:
assumes $A' \rightarrow\beta^*\ A''$ **and** $A \rightarrow\beta^*\ A'$
shows $A \rightarrow\beta^*\ A''$
⟨*proof*⟩

lemma *beta-reduces-fvs*:
assumes $A \rightarrow\beta^*\ A'$

shows *fvs* $A' \subseteq$ *fvs* A
⟨*proof*⟩

lemma *beta-reduces-app-left*:
assumes $A \rightarrow\beta^*\ A'$
shows $(App\ A\ B) \rightarrow\beta^*\ (App\ A'\ B)$
⟨*proof*⟩

lemma *beta-reduces-app-right*:
assumes $B \rightarrow\beta^*\ B'$
shows $(App\ A\ B) \rightarrow\beta^*\ (App\ A\ B')$
⟨*proof*⟩

lemma *beta-reduces-fn*:
assumes $A \rightarrow\beta^*\ A'$
shows $(Fn\ x\ T\ A) \rightarrow\beta^*\ (Fn\ x\ T\ A')$
⟨*proof*⟩

lemma *beta-reduces-pair-left*:
assumes $A \rightarrow\beta^*\ A'$
shows $(Pair\ A\ B) \rightarrow\beta^*\ (Pair\ A'\ B)$
⟨*proof*⟩

lemma *beta-reduces-pair-right*:
assumes $B \rightarrow\beta^*\ B'$
shows $(Pair\ A\ B) \rightarrow\beta^*\ (Pair\ A\ B')$
⟨*proof*⟩

lemma *beta-reduces-fst*:
assumes $P \rightarrow\beta^*\ P'$
shows $(Fst\ P) \rightarrow\beta^*\ (Fst\ P')$
⟨*proof*⟩

lemma *beta-reduces-snd*:
assumes $P \rightarrow\beta^*\ P'$
shows $(Snd\ P) \rightarrow\beta^*\ (Snd\ P')$
⟨*proof*⟩

theorem *preservation*:
assumes $M \rightarrow\beta^*\ M'\ \Gamma \vdash M : \tau$
shows $\Gamma \vdash M' : \tau$
⟨*proof*⟩

theorem *safety*:
assumes $M \rightarrow\beta^*\ M'\ \Gamma \vdash M : \tau$
shows $NF\ M' \vee (\exists M''.\ (M' \rightarrow\beta\ M''))$
⟨*proof*⟩

inductive *parallel-reduction* :: *'a trm* ⇒ *'a trm* ⇒ *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 $(\pi \cdot A) >> (\pi \cdot A')$
⟨*proof*⟩

lemma *parallel-reduction-fvs*:
assumes $A >> A'$
shows $fvs\ A' \subseteq fvs\ A$
⟨*proof*⟩

inductive-cases *parallel-reduction-unitE'*: $Unit >> A$
lemma *parallel-reduction-unitE*:
assumes $Unit >> A$
shows $A = Unit$
⟨*proof*⟩

inductive-cases *parallel-reduction-varE'*: $(Var\ x) >> A$
lemma *parallel-reduction-varE*:
assumes $(Var\ x) >> A$
shows $A = Var\ x$
⟨*proof*⟩

inductive-cases *parallel-reduction-fnE'*: $(Fn\ x\ T\ A) >> X$
lemma *parallel-reduction-fnE*:
assumes $(Fn\ x\ T\ A) >> X$
shows $X = Fn\ x\ T\ A \vee (\exists A'.\ (A >> A') \wedge X = Fn\ x\ T\ A')$
⟨*proof*⟩

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
$(X = App\ (Fn\ x\ T\ A)\ B) \vee$
$(\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge X = (A'[x ::= B'])) \vee$
$(\exists A'\ B'.\ ((Fn\ x\ T\ A) >> (Fn\ x\ T\ A')) \wedge (B >> B') \wedge X = (App\ (Fn\ x\ T\ A')\ B'))$

⟨*proof*⟩

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 $\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge X = (App\ A'\ B')$
⟨*proof*⟩

inductive-cases *parallel-reduction-pairE'*: $(Pair\ A\ B) >> X$
lemma *parallel-reduction-pairE*:
assumes $(Pair\ A\ B) >> X$
shows $\exists A'\ B'.\ (A >> A') \wedge (B >> B') \wedge (X = Pair\ A'\ B')$
⟨*proof*⟩

inductive-cases *parallel-reduction-fstE'*: $(Fst\ P) >> X$
lemma *parallel-reduction-fstE*:
assumes $(Fst\ P) >> X$
shows $(\exists P'.\ (P >> P') \wedge X = (Fst\ P')) \vee (\exists A\ A'\ B.\ (P = Pair\ A\ B) \wedge (A >> A') \wedge X = A')$
⟨*proof*⟩

inductive-cases *parallel-reduction-sndE'*: $(Snd\ P) >> X$
lemma *parallel-reduction-sndE*:
assumes $(Snd\ P) >> X$
shows $(\exists P'.\ (P >> P') \wedge X = (Snd\ P')) \vee (\exists A\ B\ B'.\ (P = Pair\ A\ B) \wedge (B >> B') \wedge X = B')$
⟨*proof*⟩

lemma *parallel-reduction-subst-inner*:
assumes $M >> M'$
shows $X[z ::= M] >> (X[z ::= M'])$
⟨*proof*⟩

lemma *parallel-reduction-subst*:
assumes $X >> X'\ M >> M'$
shows $X[z ::= M] >> (X'[z ::= M'])$
⟨*proof*⟩

inductive *complete-development* :: *'a trm* ⇒ *'a trm* ⇒ *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*⟩

lemma *not-pair-prm*:
  assumes $\bigwedge A\ B.\ P \neq Pair\ A\ B$
  shows $\pi \cdot P \neq Pair\ A\ B$
⟨*proof*⟩

lemma *complete-development-prm*:
  assumes $A >>> A'$
  shows $(\pi \cdot A) >>> (\pi \cdot A')$
⟨*proof*⟩

lemma *complete-development-fvs*:
  assumes $A >>> A'$
  shows $fvs\ A' \subseteq fvs\ A$
⟨*proof*⟩

lemma *complete-development-fnE*:
  assumes $(Fn\ x\ T\ A) >>> X$
  shows $\exists A'.\ (A >>> A') \wedge X = Fn\ x\ T\ A'$
⟨*proof*⟩

lemma *complete-development-pairE*:
  assumes $(Pair\ A\ B) >>> X$
  shows $\exists A'\ B'.\ (A >>> A') \wedge (B >>> B') \wedge X = Pair\ A'\ B'$
⟨*proof*⟩

lemma *complete-development-exists*:
  shows $\exists X.\ (A >>> X)$
⟨*proof*⟩

lemma *complete-development-triangle*:
  assumes $A >>> D\ A >> B$
  shows $B >> D$
⟨*proof*⟩

lemma *parallel-reduction-diamond*:
  assumes $A >> B\ A >> C$
  shows $\exists D.\ (B >> D) \wedge (C >> D)$
⟨*proof*⟩

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

lemma *parallel-reduces-diamond'*:

assumes $A >>^* C$ $A >> B$
shows $\exists D.\ (B >>^* D) \wedge (C >> D)$
⟨*proof*⟩

lemma *parallel-reduces-diamond*:
assumes $A >>^* B$ $A >>^* C$
shows $\exists D.\ (B >>^* D) \wedge (C >>^* D)$
⟨*proof*⟩

lemma *beta-reduction-is-parallel-reduction*:
assumes $A \rightarrow\beta B$
shows $A >> B$
⟨*proof*⟩

lemma *parallel-reduction-is-beta-reduction*:
assumes $A >> B$
shows $A \rightarrow\beta^* B$
⟨*proof*⟩

lemma *parallel-beta-reduces-equivalent*:
shows $(A >>^* B) = (A \rightarrow\beta^* B)$
⟨*proof*⟩

theorem *confluence*:
assumes $A \rightarrow\beta^* B$ $A \rightarrow\beta^* C$
shows $\exists D.\ (B \rightarrow\beta^* D) \wedge (C \rightarrow\beta^* D)$
⟨*proof*⟩

end
end