

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

Michael Rawson

December 7, 2022

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  $\implies$  fresh-in S  $\notin$  S
```

```
instantiation nat :: fresh
```

```
begin
```

```
  definition fresh-in-nat :: nat set  $\Rightarrow$  nat where
```

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

```
  instance  $\langle$ proof $\rangle$ 
```

```
end
```

```
end
```

```
theory Permutation
```

```
imports Main
```

```
begin
```

```
type-synonym 'a swp = 'a  $\times$  'a
```

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

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

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

definition *preprm-compose* :: 'a preprm \Rightarrow 'a preprm \Rightarrow 'a preprm where
preprm-compose f g \equiv f @ g

definition *preprm-unit* :: 'a \Rightarrow 'a \Rightarrow 'a preprm where
preprm-unit a b \equiv [(a, b)]

definition *preprm-ext* :: 'a preprm \Rightarrow 'a preprm \Rightarrow bool (infix =p 100) where
 $\pi =_p \sigma \equiv \forall x. \text{preprm-apply } \pi x = \text{preprm-apply } \sigma x$

definition *preprm-inv* :: 'a preprm \Rightarrow 'a preprm where
preprm-inv $\pi \equiv \text{rev } \pi$

lemma *swp-apply-unequal*:
assumes $x \neq y$
shows *swp-apply* s x \neq *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 $\{x.$

$$\text{preprm-apply } (\text{preprm-compose } (\text{preprm-unit } a \ b) \ (\text{preprm-unit } b \ c)) \ x \neq$$

$$\text{preprm-apply } (\text{preprm-unit } a \ c) \ x$$

$$\} = \{a, b\}$$
 <proof>

lemma *preprm-compose-push*:

shows

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

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

<proof>

lemma *preprm-ext-compose-left*:

assumes $P =_p S$

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

<proof>

lemma *preprm-ext-compose-right*:

assumes $P =_p S$

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

<proof>

lemma *preprm-ext-uncompose*:

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

shows $P =_p S$

<proof>

lemma *preprm-inv-compose*:

shows $\text{preprm-compose } (\text{preprm-inv } \pi) \ \pi =_p \text{preprm-id}$

<proof>

lemma *preprm-inv-involution*:

shows $\text{preprm-inv } (\text{preprm-inv } \pi) = \pi$

<proof>

lemma *preprm-inv-ext*:

assumes $\pi =_p \sigma$

shows $\text{preprm-inv } \pi =_p \text{preprm-inv } \sigma$

<proof>

quotient-type $'a \ \text{prm} = 'a \ \text{preprm} / \text{preprm-ext}$

<proof>

lift-definition $\text{prm-id} :: 'a \ \text{prm} \ (\varepsilon) \ \mathbf{is} \ \text{preprm-id}$ <proof>

lift-definition $\text{prm-apply} :: 'a \ \text{prm} \Rightarrow 'a \Rightarrow 'a \ (\mathbf{infix} \ \$ \ 140) \ \mathbf{is} \ \text{preprm-apply}$

<proof>

lift-definition $\text{prm-compose} :: 'a \ \text{prm} \Rightarrow 'a \ \text{prm} \Rightarrow 'a \ \text{prm} \ (\mathbf{infixr} \ \diamond \ 145) \ \mathbf{is}$

preprm-compose
<proof>

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

lift-definition *prm-inv* :: 'a *prm* \Rightarrow 'a *prm* **is** *preprm-inv*
<proof>

lemma *prm-apply-composition*:
 fixes *f g* :: 'a *prm* **and** *x* :: 'a
 shows $f \diamond g \$ x = f \$ (g \$ x)$
<proof>

lemma *prm-apply-unequal*:
 fixes π :: 'a *prm* **and** *x y* :: 'a
 assumes $x \neq y$
 shows $\pi \$ x \neq \pi \$ y$
<proof>

lemma *prm-unit-equal-id*:
 fixes *a* :: 'a
 shows $[a \leftrightarrow a] = \varepsilon$
<proof>

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

lemma *prm-unit-action*:
 fixes *a b* :: 'a
 shows $[a \leftrightarrow b] \$ a = b$
<proof>

lemma *prm-unit-commutes*:
 fixes *a b* :: 'a
 shows $[a \leftrightarrow b] = [b \leftrightarrow a]$
<proof>

lemma *prm-unit-involution*:
 fixes *a b* :: 'a
 shows $[a \leftrightarrow b] \diamond [a \leftrightarrow b] = \varepsilon$
<proof>

lemma *prm-apply-id*:
 fixes *x* :: 'a
 shows $\varepsilon \$ x = x$
<proof>

lemma *prm-apply-injective*:

shows $\text{inj } (\text{prm-apply } \pi)$

<proof>

lemma *prm-inv-compose*:

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

<proof>

interpretation *'a prm: semigroup prm-compose*

<proof>

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

<proof>

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

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

lemma *prm-set-apply-compose*:

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

<proof>

lemma *prm-set-membership*:

assumes $x \in S$

shows $\pi \$ x \in \pi \{\$\} S$

<proof>

lemma *prm-set-notmembership*:

assumes $x \notin S$

shows $\pi \$ x \notin \pi \{\$\} S$

<proof>

lemma *prm-set-singleton*:

shows $\pi \{\$\} \{x\} = \{\pi \$ x\}$

<proof>

lemma *prm-set-id*:

shows $\varepsilon \{\$\} S = S$

<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 = $\{\}$

| $\text{ptrm-fvs } (P\text{Var } x) = \{x\}$
 | $\text{ptrm-fvs } (P\text{App } A B) = \text{ptrm-fvs } A \cup \text{ptrm-fvs } B$
 | $\text{ptrm-fvs } (P\text{Fn } x \text{ - } A) = \text{ptrm-fvs } A - \{x\}$
 | $\text{ptrm-fvs } (P\text{Pair } A B) = \text{ptrm-fvs } A \cup \text{ptrm-fvs } B$
 | $\text{ptrm-fvs } (P\text{Fst } P) = \text{ptrm-fvs } P$
 | $\text{ptrm-fvs } (P\text{Snd } P) = \text{ptrm-fvs } P$

fun $\text{ptrm-apply-prm} :: 'a \text{ prm} \Rightarrow 'a \text{ ptrm} \Rightarrow 'a \text{ ptrm}$ (**infixr** \cdot 150) **where**
 $\text{ptrm-apply-prm } \pi \text{ PUnit} = \text{PUnit}$
 | $\text{ptrm-apply-prm } \pi (P\text{Var } x) = P\text{Var } (\pi \$ x)$
 | $\text{ptrm-apply-prm } \pi (P\text{App } A B) = P\text{App } (\text{ptrm-apply-prm } \pi A) (\text{ptrm-apply-prm } \pi B)$
 | $\text{ptrm-apply-prm } \pi (P\text{Fn } x \text{ T } A) = P\text{Fn } (\pi \$ x) \text{ T } (\text{ptrm-apply-prm } \pi A)$
 | $\text{ptrm-apply-prm } \pi (P\text{Pair } A B) = P\text{Pair } (\text{ptrm-apply-prm } \pi A) (\text{ptrm-apply-prm } \pi B)$
 | $\text{ptrm-apply-prm } \pi (P\text{Fst } P) = P\text{Fst } (\text{ptrm-apply-prm } \pi P)$
 | $\text{ptrm-apply-prm } \pi (P\text{Snd } P) = P\text{Snd } (\text{ptrm-apply-prm } \pi P)$

inductive $\text{ptrm-alpha-equiv} :: 'a \text{ ptrm} \Rightarrow 'a \text{ ptrm} \Rightarrow \text{bool}$ (**infix** \approx 100) **where**
 $\text{unit: } P\text{Unit} \approx P\text{Unit}$
 | $\text{var: } (P\text{Var } x) \approx (P\text{Var } x)$
 | $\text{app: } \llbracket A \approx B; C \approx D \rrbracket \Longrightarrow (P\text{App } A C) \approx (P\text{App } B D)$
 | $\text{fn1: } A \approx B \Longrightarrow (P\text{Fn } x \text{ T } A) \approx (P\text{Fn } x \text{ T } B)$
 | $\text{fn2: } \llbracket a \neq b; A \approx [a \leftrightarrow b] \cdot B; a \notin \text{ptrm-fvs } B \rrbracket \Longrightarrow (P\text{Fn } a \text{ T } A) \approx (P\text{Fn } b \text{ T } B)$
 | $\text{pair: } \llbracket A \approx B; C \approx D \rrbracket \Longrightarrow (P\text{Pair } A C) \approx (P\text{Pair } B D)$
 | $\text{fst: } A \approx B \Longrightarrow P\text{Fst } A \approx P\text{Fst } B$
 | $\text{snd: } A \approx B \Longrightarrow P\text{Snd } A \approx P\text{Snd } B$

inductive-cases $\text{unitE: } P\text{Unit} \approx Y$
inductive-cases $\text{varE: } P\text{Var } x \approx Y$
inductive-cases $\text{appE: } P\text{App } A B \approx Y$
inductive-cases $\text{fnE: } P\text{Fn } x \text{ T } A \approx Y$
inductive-cases $\text{pairE: } P\text{Pair } A B \approx Y$
inductive-cases $\text{fstE: } P\text{Fst } P \approx Y$
inductive-cases $\text{sndE: } P\text{Snd } P \approx Y$

lemma $\text{ptrm-prm-apply-id:}$

shows $\varepsilon \cdot X = X$

<proof>

lemma $\text{ptrm-prm-apply-compose:}$

shows $\pi \cdot \sigma \cdot X = (\pi \diamond \sigma) \cdot X$

<proof>

lemma ptrm-size-prm:

shows $\text{size } X = \text{size } (\pi \cdot X)$

<proof>

lemma *ptrm-size-alpha-equiv*:

assumes $X \approx Y$

shows $\text{size } X = \text{size } Y$

<proof>

lemma *ptrm-fvs-finite*:

shows $\text{finite } (\text{ptrm-fvs } X)$

<proof>

lemma *ptrm-prm-fvs*:

shows $\text{ptrm-fvs } (\pi \cdot X) = \pi \{ \$ \} \text{ ptrm-fvs } X$

<proof>

lemma *ptrm-alpha-equiv-fvs*:

assumes $X \approx Y$

shows $\text{ptrm-fvs } X = \text{ptrm-fvs } Y$

<proof>

lemma *ptrm-alpha-equiv-prm*:

assumes $X \approx Y$

shows $\pi \cdot X \approx \pi \cdot Y$

<proof>

lemma *ptrm-swp-transfer*:

shows $[a \leftrightarrow b] \cdot X \approx Y \longleftrightarrow X \approx [a \leftrightarrow b] \cdot Y$

<proof>

lemma *ptrm-alpha-equiv-fvs-transfer*:

assumes $A \approx [a \leftrightarrow b] \cdot B$ and $a \notin \text{ptrm-fvs } B$

shows $b \notin \text{ptrm-fvs } A$

<proof>

lemma *ptrm-prm-agreement-equiv*:

assumes $\bigwedge a. a \in \text{ds } \pi \sigma \implies a \notin \text{ptrm-fvs } M$

shows $\pi \cdot M \approx \sigma \cdot M$

<proof>

lemma *ptrm-prm-unit-inaction*:

assumes $a \notin \text{ptrm-fvs } X$ and $b \notin \text{ptrm-fvs } X$

shows $[a \leftrightarrow b] \cdot X \approx X$

<proof>

lemma *ptrm-alpha-equiv-reflexive*:

shows $M \approx M$

<proof>

corollary *ptrm-alpha-equiv-reflp*:

shows $\text{reflp } \text{ptrm-alpha-equiv}$

<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 \rightarrow 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) = 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') \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 $\text{ptrm-infer-type } (\Gamma(a \mapsto T, b \mapsto S)) X = \text{ptrm-infer-type } (\Gamma(a \mapsto S, b \mapsto T)) ([a \leftrightarrow b] \cdot X)$
 $\langle \text{proof} \rangle$

lemma $\text{ptrm-infer-type-swp}$:

assumes $a \neq b \ b \notin \text{ptrm-fvs } X$
shows $\text{ptrm-infer-type } (\Gamma(a \mapsto \tau)) X = \text{ptrm-infer-type } (\Gamma(b \mapsto \tau)) ([a \leftrightarrow b] \cdot X)$
 $\langle \text{proof} \rangle$

lemma $\text{ptrm-infer-type-alpha-equiv}$:

assumes $X \approx Y$
shows $\text{ptrm-infer-type } \Gamma X = \text{ptrm-infer-type } \Gamma Y$
 $\langle \text{proof} \rangle$

end

theory SimplyTyped

imports PreSimplyTyped

begin

quotient-type $'a \text{ trm} = 'a \text{ ptrm} / \text{ptrm-alpha-equiv}$
 $\langle \text{proof} \rangle$

lift-definition $\text{Unit} :: 'a \text{ trm} \text{ is } P\text{Unit} \langle \text{proof} \rangle$

lift-definition $\text{Var} :: 'a \Rightarrow 'a \text{ trm} \text{ is } P\text{Var} \langle \text{proof} \rangle$

lift-definition $\text{App} :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ is } P\text{App} \langle \text{proof} \rangle$

lift-definition $\text{Fn} :: 'a \Rightarrow \text{type} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ is } P\text{Fn} \langle \text{proof} \rangle$

lift-definition $\text{Pair} :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ is } P\text{Pair} \langle \text{proof} \rangle$

lift-definition $\text{Fst} :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ is } P\text{Fst} \langle \text{proof} \rangle$

lift-definition $\text{Snd} :: 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ is } P\text{Snd} \langle \text{proof} \rangle$

lift-definition $\text{fvs} :: 'a \text{ trm} \Rightarrow 'a \text{ set} \text{ is } \text{ptrm-fvs} \langle \text{proof} \rangle$

lift-definition $\text{prm} :: 'a \text{ prm} \Rightarrow 'a \text{ trm} \Rightarrow 'a \text{ trm} \text{ (infixr } \cdot 150) \text{ is } \text{ptrm-apply-prm}$
 $\langle \text{proof} \rangle$

lift-definition $\text{depth} :: 'a \text{ trm} \Rightarrow \text{nat} \text{ is } \text{size} \langle \text{proof} \rangle$

lemma depth-prm :

shows $\text{depth } (\pi \cdot A) = \text{depth } A$
 $\langle \text{proof} \rangle$

lemma depth-app :

shows $\text{depth } A < \text{depth } (\text{App } A B) \ \text{depth } B < \text{depth } (\text{App } A B)$
 $\langle \text{proof} \rangle$

lemma depth-fn :

shows $\text{depth } A < \text{depth } (\text{Fn } x T A)$
 $\langle \text{proof} \rangle$

lemma depth-pair :

shows $\text{depth } A < \text{depth } (\text{Pair } A \ B) \ \text{depth } B < \text{depth } (\text{Pair } A \ B)$
(proof)

lemma *depth-fst*:
shows $\text{depth } P < \text{depth } (\text{Fst } P)$
(proof)

lemma *depth-snd*:
shows $\text{depth } P < \text{depth } (\text{Snd } P)$
(proof)

lemma *unit-not-var*:
shows $\text{Unit} \neq \text{Var } x$
(proof)

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

lemma *unit-not-fn*:
shows $\text{Unit} \neq \text{Fn } x \ T \ A$
(proof)

lemma *unit-not-pair*:
shows $\text{Unit} \neq \text{Pair } A \ B$
(proof)

lemma *unit-not-fst*:
shows $\text{Unit} \neq \text{Fst } P$
(proof)

lemma *unit-not-snd*:
shows $\text{Unit} \neq \text{Snd } P$
(proof)

lemma *var-not-app*:
shows $\text{Var } x \neq \text{App } A \ B$
(proof)

lemma *var-not-fn*:
shows $\text{Var } x \neq \text{Fn } y \ T \ A$
(proof)

lemma *var-not-pair*:
shows $\text{Var } x \neq \text{Pair } A \ B$
(proof)

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

<proof>

lemma *var-not-snd:*

shows $\text{Var } x \neq \text{Snd } P$

<proof>

lemma *app-not-fn:*

shows $\text{App } A B \neq \text{Fn } y T X$

<proof>

lemma *app-not-pair:*

shows $\text{App } A B \neq \text{Pair } C D$

<proof>

lemma *app-not-fst:*

shows $\text{App } A B \neq \text{Fst } P$

<proof>

lemma *app-not-snd:*

shows $\text{App } A B \neq \text{Snd } P$

<proof>

lemma *fn-not-pair:*

shows $\text{Fn } x T A \neq \text{Pair } C D$

<proof>

lemma *fn-not-fst:*

shows $\text{Fn } x T A \neq \text{Fst } P$

<proof>

lemma *fn-not-snd:*

shows $\text{Fn } x T A \neq \text{Snd } P$

<proof>

lemma *pair-not-fst:*

shows $\text{Pair } A B \neq \text{Fst } P$

<proof>

lemma *pair-not-snd:*

shows $\text{Pair } A B \neq \text{Snd } P$

<proof>

lemma *fst-not-snd:*

shows $\text{Fst } P \neq \text{Snd } Q$

<proof>

lemma *trm-simp:*

shows

$\text{Var } x = \text{Var } y \implies x = y$

$$\begin{aligned}
& App\ A\ B = App\ C\ D \implies A = C \\
& App\ A\ B = App\ C\ D \implies B = D \\
& Fn\ x\ T\ A = Fn\ y\ S\ B \implies \\
& \quad (x = y \wedge T = S \wedge A = B) \vee (x \neq y \wedge T = S \wedge x \notin fvs\ B \wedge A = [x \leftrightarrow y] \cdot
\end{aligned}$$

B)

$$\begin{aligned}
& Pair\ A\ B = Pair\ C\ D \implies A = C \\
& Pair\ A\ B = Pair\ C\ D \implies B = D \\
& Fst\ P = Fst\ Q \implies P = Q \\
& Snd\ P = Snd\ Q \implies P = Q
\end{aligned}$$

$\langle proof \rangle$

lemma *fn-eq*:

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

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

$\langle proof \rangle$

lemma *trm-prm-simp*:

shows

$$\begin{aligned}
& \pi \cdot Unit = Unit \\
& \pi \cdot Var\ x = Var\ (\pi \$ x) \\
& \pi \cdot App\ A\ B = App\ (\pi \cdot A)\ (\pi \cdot B) \\
& \pi \cdot Fn\ x\ T\ A = Fn\ (\pi \$ x)\ T\ (\pi \cdot A) \\
& \pi \cdot Pair\ A\ B = Pair\ (\pi \cdot A)\ (\pi \cdot B) \\
& \pi \cdot Fst\ P = Fst\ (\pi \cdot P) \\
& \pi \cdot Snd\ P = Snd\ (\pi \cdot P)
\end{aligned}$$

$\langle proof \rangle$

lemma *trm-prm-apply-compose*:

shows $\pi \cdot \sigma \cdot A = (\pi \diamond \sigma) \cdot A$

$\langle proof \rangle$

lemma *fvs-finite*:

shows *finite* (*fvs* *M*)

$\langle proof \rangle$

lemma *fvs-simp*:

shows

$$\begin{aligned}
& fvs\ Unit = \{\} \text{ and} \\
& fvs\ (Var\ x) = \{x\} \\
& fvs\ (App\ A\ B) = fvs\ A \cup fvs\ B \\
& fvs\ (Fn\ x\ T\ A) = fvs\ A - \{x\} \\
& fvs\ (Pair\ A\ B) = fvs\ A \cup fvs\ B \\
& fvs\ (Fst\ P) = fvs\ P \\
& fvs\ (Snd\ P) = fvs\ P
\end{aligned}$$

$\langle proof \rangle$

lemma *var-prm-action*:

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

$\langle proof \rangle$

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

lemma *trm-prm-apply-id*:
shows $\varepsilon \cdot M = M$
 $\langle \text{proof} \rangle$

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

lemma *trm-prm-agreement-equiv*:
assumes $\bigwedge a. a \in \text{ds } \pi \ \sigma \implies a \notin \text{fvs } M$
shows $\pi \cdot M = \sigma \cdot M$
 $\langle \text{proof} \rangle$

lemma *trm-induct*:
fixes $P :: 'a \text{ trm} \Rightarrow \text{bool}$
assumes
 $P \ \text{Unit}$
 $\bigwedge x. P \ (\text{Var } x)$
 $\bigwedge A \ B. \llbracket P \ A; P \ B \rrbracket \implies P \ (\text{App } A \ B)$
 $\bigwedge x \ T \ A. P \ A \implies P \ (\text{Fn } x \ T \ A)$
 $\bigwedge A \ B. \llbracket P \ A; P \ B \rrbracket \implies P \ (\text{Pair } A \ B)$
 $\bigwedge A. P \ A \implies P \ (\text{Fst } A)$
 $\bigwedge A. P \ A \implies P \ (\text{Snd } A)$
shows $P \ M$
 $\langle \text{proof} \rangle$

lemma *trm-cases*:
assumes
 $M = \text{Unit} \implies P \ M$
 $\bigwedge x. M = \text{Var } x \implies P \ M$
 $\bigwedge A \ B. M = \text{App } A \ B \implies P \ M$
 $\bigwedge x \ T \ A. M = \text{Fn } x \ T \ A \implies P \ M$
 $\bigwedge A \ B. M = \text{Pair } A \ B \implies P \ M$
 $\bigwedge A. M = \text{Fst } A \implies P \ M$
 $\bigwedge A. M = \text{Snd } A \implies P \ M$
shows $P \ M$
 $\langle \text{proof} \rangle$

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

$\bigwedge A B. \llbracket \bigwedge K. \text{depth } K < \text{depth } (\text{App } A B) \implies P K \rrbracket \implies P (\text{App } A B)$
 $\bigwedge M x T A. (\bigwedge K. \text{depth } K < \text{depth } (\text{Fn } x T A) \implies P K) \implies P (\text{Fn } x T A)$
 $\bigwedge A B. \llbracket \bigwedge K. \text{depth } K < \text{depth } (\text{Pair } A B) \implies P K \rrbracket \implies P (\text{Pair } A B)$
 $\bigwedge A. \llbracket \bigwedge K. \text{depth } K < \text{depth } (\text{Fst } A) \implies P K \rrbracket \implies P (\text{Fst } A)$
 $\bigwedge A. \llbracket \bigwedge K. \text{depth } K < \text{depth } (\text{Snd } A) \implies P K \rrbracket \implies P (\text{Snd } A)$

shows $P M$

$\langle \text{proof} \rangle$

context *fresh begin*

lemma *fresh-fn*:

fixes $x :: 'a$ **and** $S :: 'a \text{ set}$

assumes *finite S*

shows $\exists y B. y \notin S \wedge B = [y \leftrightarrow x] \cdot A \wedge (\text{Fn } x T A = \text{Fn } y T B)$

$\langle \text{proof} \rangle$

lemma *trm-strong-induct*:

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

assumes

$P S \text{ Unit}$

$\bigwedge x. P S (\text{Var } x)$

$\bigwedge A B. \llbracket P S A; P S B \rrbracket \implies P S (\text{App } A B)$

$\bigwedge x T A. x \notin S \implies (\bigwedge A. P S A \implies P S (\text{Fn } x T A))$

$\bigwedge A B. \llbracket P S A; P S B \rrbracket \implies P S (\text{Pair } A B)$

$\bigwedge A. P S A \implies P S (\text{Fst } A)$

$\bigwedge A. P S A \implies P S (\text{Snd } A)$

finite S

shows $P S M$

$\langle \text{proof} \rangle$

lemma *trm-strong-cases*:

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

assumes

$M = \text{Unit} \implies P S M$

$\bigwedge x. M = \text{Var } x \implies P S M$

$\bigwedge A B. M = \text{App } A B \implies P S M$

$\bigwedge x T A. M = \text{Fn } x T A \implies x \notin S \implies P S M$

$\bigwedge A B. M = \text{Pair } A B \implies P S M$

$\bigwedge A. M = \text{Fst } A \implies P S M$

$\bigwedge A. M = \text{Snd } A \implies P S M$

finite S

shows $P S M$

$\langle \text{proof} \rangle$

lemma *trm-strong-depth-induct*:

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

assumes

$P S \text{ Unit}$

$\bigwedge x. P S (\text{Var } x)$

$\wedge A B. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{App } A B) \implies P S K \rrbracket \implies P S (\text{App } A B)$
 $\wedge x T. x \notin S \implies (\wedge A. (\wedge K. \text{depth } K < \text{depth } (\text{Fn } x T A) \implies P S K) \implies P S (\text{Fn } x T A))$
 $\wedge A B. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Pair } A B) \implies P S K \rrbracket \implies P S (\text{Pair } A B)$
 $\wedge A. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Fst } A) \implies P S K \rrbracket \implies P S (\text{Fst } A)$
 $\wedge A. \llbracket \wedge K. \text{depth } K < \text{depth } (\text{Snd } A) \implies P S K \rrbracket \implies P S (\text{Snd } A)$
finite S
shows $P S M$
 $\langle \text{proof} \rangle$

lemma *trm-prm-fvs*:
shows $\text{fvs } (\pi \cdot M) = \pi \{ \$ \} \text{fvs } M$
 $\langle \text{proof} \rangle$

inductive *typing* :: 'a *typing-ctx* \Rightarrow 'a *trm* \Rightarrow *type* \Rightarrow *bool* ($- \vdash - : -$) **where**
tunit: $\Gamma \vdash \text{Unit} : T\text{Unit}$
tvar: $\Gamma x = \text{Some } \tau \implies \Gamma \vdash \text{Var } x : \tau$
tapp: $\llbracket \Gamma \vdash f : (T\text{Arr } \tau \sigma); \Gamma \vdash x : \tau \rrbracket \implies \Gamma \vdash \text{App } f x : \sigma$
tfn: $\Gamma(x \mapsto \tau) \vdash A : \sigma \implies \Gamma \vdash \text{Fn } x \tau A : (T\text{Arr } \tau \sigma)$
tpair: $\llbracket \Gamma \vdash A : \tau; \Gamma \vdash B : \sigma \rrbracket \implies \Gamma \vdash \text{Pair } A B : (T\text{Pair } \tau \sigma)$
tfst: $\Gamma \vdash P : (T\text{Pair } \tau \sigma) \implies \Gamma \vdash \text{Fst } P : \tau$
tsnd: $\Gamma \vdash P : (T\text{Pair } \tau \sigma) \implies \Gamma \vdash \text{Snd } P : \sigma$

lemma *typing-prm*:
assumes $\Gamma \vdash M : \tau \wedge y. y \in \text{fvs } M \implies \Gamma y = \Delta (\pi \$ y)$
shows $\Delta \vdash \pi \cdot M : \tau$
 $\langle \text{proof} \rangle$

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

lemma *typing-unitE*:
assumes $\Gamma \vdash \text{Unit} : \tau$
shows $\tau = T\text{Unit}$
 $\langle \text{proof} \rangle$

lemma *typing-varE*:
assumes $\Gamma \vdash \text{Var } x : \tau$
shows $\Gamma x = \text{Some } \tau$
 $\langle \text{proof} \rangle$

lemma *typing-appE*:
assumes $\Gamma \vdash \text{App } A B : \sigma$
shows $\exists \tau. (\Gamma \vdash A : (T\text{Arr } \tau \sigma)) \wedge (\Gamma \vdash B : \tau)$
 $\langle \text{proof} \rangle$

lemma *typing-fnE*:

assumes $\Gamma \vdash Fn\ x\ T\ A : \vartheta$
shows $\exists \sigma. \vartheta = (TArr\ T\ \sigma) \wedge (\Gamma(x \mapsto T) \vdash A : \sigma)$
 $\langle proof \rangle$

lemma *typing-pairE*:
assumes $\Gamma \vdash Pair\ A\ B : \vartheta$
shows $\exists \tau\ \sigma. \vartheta = (TPair\ \tau\ \sigma) \wedge (\Gamma \vdash A : \tau) \wedge (\Gamma \vdash B : \sigma)$
 $\langle proof \rangle$

lemma *typing-fstE*:
assumes $\Gamma \vdash Fst\ P : \tau$
shows $\exists \sigma. (\Gamma \vdash P : (TPair\ \tau\ \sigma))$
 $\langle proof \rangle$

lemma *typing-sndE*:
assumes $\Gamma \vdash Snd\ P : \sigma$
shows $\exists \tau. (\Gamma \vdash P : (TPair\ \tau\ \sigma))$
 $\langle proof \rangle$

theorem *typing-weaken*:
assumes $\Gamma \vdash M : \tau\ y \notin fvs\ M$
shows $\Gamma(y \mapsto \sigma) \vdash M : \tau$
 $\langle proof \rangle$

lift-definition *infer* :: 'a *typing-ctx* \Rightarrow 'a *trm* \Rightarrow *type option* **is** *ptrm-infer-type*
 $\langle proof \rangle$

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

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

$$\begin{aligned} &) \\ & \text{infer } \Gamma \text{ (Snd } P) = (\text{case infer } \Gamma \text{ } P \text{ of} \\ & \quad (\text{Some (TPair } \tau \sigma)) \Rightarrow \text{Some } \sigma \\ & \quad | - \Rightarrow \text{None} \\ &) \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *infer-unitE*:
assumes $\text{infer } \Gamma \text{ Unit} = \text{Some } \tau$
shows $\tau = \text{TUnit}$
 $\langle \text{proof} \rangle$

lemma *infer-varE*:
assumes $\text{infer } \Gamma \text{ (Var } x) = \text{Some } \tau$
shows $\Gamma \text{ } x = \text{Some } \tau$
 $\langle \text{proof} \rangle$

lemma *infer-appE*:
assumes $\text{infer } \Gamma \text{ (App } A \text{ } B) = \text{Some } \tau$
shows $\exists \sigma. \text{infer } \Gamma \text{ } A = \text{Some (TArr } \sigma \tau) \wedge \text{infer } \Gamma \text{ } B = \text{Some } \sigma$
 $\langle \text{proof} \rangle$

lemma *infer-fnE*:
assumes $\text{infer } \Gamma \text{ (Fn } x \text{ } T \text{ } A) = \text{Some } \tau$
shows $\exists \sigma. \tau = \text{TArr } T \sigma \wedge \text{infer } (\Gamma(x \mapsto T)) \text{ } A = \text{Some } \sigma$
 $\langle \text{proof} \rangle$

lemma *infer-pairE*:
assumes $\text{infer } \Gamma \text{ (Pair } A \text{ } B) = \text{Some } \tau$
shows $\exists T \text{ } S. \tau = \text{TPair } T \text{ } S \wedge \text{infer } \Gamma \text{ } A = \text{Some } T \wedge \text{infer } \Gamma \text{ } B = \text{Some } S$
 $\langle \text{proof} \rangle$

lemma *infer-fstE*:
assumes $\text{infer } \Gamma \text{ (Fst } P) = \text{Some } \tau$
shows $\exists T \text{ } S. \text{infer } \Gamma \text{ } P = \text{Some (TPair } T \text{ } S) \wedge \tau = T$
 $\langle \text{proof} \rangle$

lemma *infer-sndE*:
assumes $\text{infer } \Gamma \text{ (Snd } P) = \text{Some } \tau$
shows $\exists T \text{ } S. \text{infer } \Gamma \text{ } P = \text{Some (TPair } T \text{ } S) \wedge \tau = S$
 $\langle \text{proof} \rangle$

lemma *infer-sound*:
assumes $\text{infer } \Gamma \text{ } M = \text{Some } \tau$
shows $\Gamma \vdash M : \tau$
 $\langle \text{proof} \rangle$

lemma *infer-complete*:
assumes $\Gamma \vdash M : \tau$

shows $\text{infer } \Gamma M = \text{Some } \tau$
 $\langle \text{proof} \rangle$

theorem *infer-valid*:

shows $(\Gamma \vdash M : \tau) = (\text{infer } \Gamma M = \text{Some } \tau)$
 $\langle \text{proof} \rangle$

inductive *substitutes* :: 'a trm \Rightarrow 'a \Rightarrow 'a trm \Rightarrow 'a trm \Rightarrow bool **where**

unit: $\text{substitutes } \text{Unit } y M \text{Unit}$
| *var1*: $x = y \Longrightarrow \text{substitutes } (\text{Var } x) y M M$
| *var2*: $x \neq y \Longrightarrow \text{substitutes } (\text{Var } x) y M (\text{Var } x)$
| *app*: $\llbracket \text{substitutes } A x M A'; \text{substitutes } B x M B' \rrbracket \Longrightarrow \text{substitutes } (\text{App } A B) x M (\text{App } A' B')$
| *fn*: $\llbracket x \neq y; y \notin \text{fvs } M; \text{substitutes } A x M A' \rrbracket \Longrightarrow \text{substitutes } (\text{Fn } y T A) x M (\text{Fn } y T A')$
| *pair*: $\llbracket \text{substitutes } A x M A'; \text{substitutes } B x M B' \rrbracket \Longrightarrow \text{substitutes } (\text{Pair } A B) x M (\text{Pair } A' B')$
| *fst*: $\text{substitutes } P x M P' \Longrightarrow \text{substitutes } (\text{Fst } P) x M (\text{Fst } P')$
| *snd*: $\text{substitutes } P x M P' \Longrightarrow \text{substitutes } (\text{Snd } P) x M (\text{Snd } P')$

lemma *substitutes-prm*:

assumes $\text{substitutes } A x M A'$
shows $\text{substitutes } (\pi \cdot A) (\pi \$ x) (\pi \cdot M) (\pi \cdot A')$
 $\langle \text{proof} \rangle$

lemma *substitutes-fvs*:

assumes $\text{substitutes } A x M A'$
shows $\text{fvs } A' \subseteq \text{fvs } A - \{x\} \cup \text{fvs } M$
 $\langle \text{proof} \rangle$

inductive-cases *substitutes-unitE'*: $\text{substitutes } \text{Unit } y M X$

lemma *substitutes-unitE*:

assumes $\text{substitutes } \text{Unit } y M X$
shows $X = \text{Unit}$
 $\langle \text{proof} \rangle$

inductive-cases *substitutes-varE'*: $\text{substitutes } (\text{Var } x) y M X$

lemma *substitutes-varE*:

assumes $\text{substitutes } (\text{Var } x) y M X$
shows $(x = y \wedge M = X) \vee (x \neq y \wedge X = \text{Var } x)$
 $\langle \text{proof} \rangle$

inductive-cases *substitutes-appE'*: $\text{substitutes } (\text{App } A B) x M X$

lemma *substitutes-appE*:

assumes $\text{substitutes } (\text{App } A B) x M X$
shows $\exists A' B'. \text{substitutes } A x M A' \wedge \text{substitutes } B x M B' \wedge X = \text{App } A' B'$
 $\langle \text{proof} \rangle$

inductive-cases *substitutes-fnE'*: $\text{substitutes } (\text{Fn } y T A) x M X$

lemma *substitutes-fnE*:

assumes *substitutes* (Fn y T A) x M X y ≠ x y ∉ fvs M

shows ∃ A'. *substitutes* A x M A' ∧ X = Fn y T A'

⟨*proof*⟩

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

lemma *substitutes-pairE*:

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

shows ∃ A' B'. *substitutes* A x M A' ∧ *substitutes* B x M B' ∧ 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 ∃ P'. *substitutes* P x M P' ∧ 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 ∃ P'. *substitutes* P x M P' ∧ X = Snd P'

⟨*proof*⟩

lemma *substitutes-total*:

shows ∃ 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 ∃! X. *substitutes* A x M X

⟨*proof*⟩

definition *subst* :: 'a trm ⇒ 'a ⇒ 'a trm ⇒ 'a trm (-[- ::= -]) **where**

subst A x M ≡ (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 ≠ y

shows $(\text{Var } x)[y ::= M] = \text{Var } x$
<proof>

lemma *subst-simp-app*:

shows $(\text{App } A \ B)[x ::= M] = \text{App } (A[x ::= M]) \ (B[x ::= M])$
<proof>

lemma *subst-simp-fn*:

assumes $x \neq y \ y \notin \text{fvs } M$
shows $(\text{Fn } y \ T \ A)[x ::= M] = \text{Fn } y \ T \ (A[x ::= M])$
<proof>

lemma *subst-simp-pair*:

shows $(\text{Pair } A \ B)[x ::= M] = \text{Pair } (A[x ::= M]) \ (B[x ::= M])$
<proof>

lemma *subst-simp-fst*:

shows $(\text{Fst } P)[x ::= M] = \text{Fst } (P[x ::= M])$
<proof>

lemma *subst-simp-snd*:

shows $(\text{Snd } P)[x ::= M] = \text{Snd } (P[x ::= M])$
<proof>

lemma *subst-prm*:

shows $(\pi \cdot (M[z ::= N])) = ((\pi \cdot M)[\pi \$ z ::= \pi \cdot N])$
<proof>

lemma *subst-fvs*:

shows $\text{fvs } (M[z ::= N]) \subseteq (\text{fvs } M - \{z\}) \cup \text{fvs } N$
<proof>

lemma *subst-free*:

assumes $y \notin \text{fvs } M$
shows $M[y ::= N] = M$
<proof>

lemma *subst-swp*:

assumes $y \notin \text{fvs } A$
shows $([y \leftrightarrow z] \cdot A)[y ::= M] = (A[z ::= M])$
<proof>

lemma *barendregt*:

assumes $y \neq z \ y \notin \text{fvs } L$
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 \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 $fvs M' \subseteq fvs M$
 ⟨proof⟩

lemma beta-reduction-prm:

assumes $M \rightarrow\beta M'$
shows $(\pi \cdot M) \rightarrow\beta (\pi \cdot M')$
 ⟨proof⟩

lemma beta-reduction-left-unitE:

assumes $Unit \rightarrow\beta M'$
shows *False*
 ⟨proof⟩

lemma beta-reduction-left-varE:

assumes $(Var x) \rightarrow\beta M'$
shows *False*
 ⟨proof⟩

lemma beta-reduction-left-appE:

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

⟨proof⟩

lemma beta-reduction-left-fnE:

assumes $(Fn x T A) \rightarrow\beta M'$
shows $\exists A'. (A \rightarrow\beta A') \wedge M' = (Fn x T A')$
 ⟨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*: $\llbracket A \neq Fn\ x\ T\ A'; NF\ A; NF\ B \rrbracket \Longrightarrow NF\ (App\ A\ B)$

| *fn*: $NF\ A \Longrightarrow NF\ (Fn\ x\ T\ A)$

| *pair*: $\llbracket NF\ A; NF\ B \rrbracket \Longrightarrow NF\ (Pair\ A\ B)$

| *fst*: $\llbracket P \neq Pair\ A\ B; NF\ P \rrbracket \Longrightarrow NF\ (Fst\ P)$

| *snd*: $\llbracket P \neq Pair\ A\ B; NF\ P \rrbracket \Longrightarrow NF\ (Snd\ P)$

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*: $\llbracket M \rightarrow\beta^* M'; M' \rightarrow\beta\ M'' \rrbracket \Longrightarrow M \rightarrow\beta^* M''$

lemma *beta-reduces-composition*:

assumes $A' \rightarrow\beta^* A''$ **and** $A \rightarrow\beta^* A'$

shows $A \rightarrow\beta^* A''$

<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 \Rightarrow 'a trm \Rightarrow bool (- >> -) **where**

refl: $A >> A$

| *beta*: $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (App (Fn x T A) B) \gg (A'[x ::= B'])$
 | *eta*: $A \gg A' \Longrightarrow (Fn x T A) \gg (Fn x T A')$
 | *app*: $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (App A B) \gg (App A' B')$
 | *pair*: $\llbracket A \gg A'; B \gg B' \rrbracket \Longrightarrow (Pair A B) \gg (Pair A' B')$
 | *fst1*: $P \gg P' \Longrightarrow (Fst P) \gg (Fst P')$
 | *fst2*: $A \gg A' \Longrightarrow (Fst (Pair A B)) \gg A'$
 | *snd1*: $P \gg P' \Longrightarrow (Snd P) \gg (Snd P')$
 | *snd2*: $B \gg B' \Longrightarrow (Snd (Pair A B)) \gg B'$

lemma *parallel-reduction-prm*:

assumes $A \gg A'$
 shows $(\pi \cdot A) \gg (\pi \cdot A')$

<proof>

lemma *parallel-reduction-fvs*:

assumes $A \gg A'$
 shows $fvs A' \subseteq fvs A$

<proof>

inductive-cases *parallel-reduction-unitE'*: $Unit \gg A$

lemma *parallel-reduction-unitE*:

assumes $Unit \gg A$
 shows $A = Unit$

<proof>

inductive-cases *parallel-reduction-varE'*: $(Var x) \gg A$

lemma *parallel-reduction-varE*:

assumes $(Var x) \gg A$
 shows $A = Var x$

<proof>

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

lemma *parallel-reduction-fnE*:

assumes $(Fn x T A) \gg X$
 shows $X = Fn x T A \vee (\exists A'. (A \gg A') \wedge X = Fn x T A')$

<proof>

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

lemma *parallel-reduction-redexE*:

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

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

<proof>

inductive-cases *parallel-reduction-nonredexE'*: $(App A B) \gg X$

lemma *parallel-reduction-nonredexE*:

assumes $(App\ A\ B) \gg X$ **and** $\bigwedge x\ T\ A'.\ A \neq Fn\ x\ T\ A'$
shows $\exists A'\ B'.\ (A \gg A') \wedge (B \gg B') \wedge X = (App\ A'\ B')$
 $\langle proof \rangle$

inductive-cases *parallel-reduction-pairE'*: $(Pair\ A\ B) \gg X$

lemma *parallel-reduction-pairE*:

assumes $(Pair\ A\ B) \gg X$
shows $\exists A'\ B'.\ (A \gg A') \wedge (B \gg B') \wedge (X = Pair\ A'\ B')$
 $\langle proof \rangle$

inductive-cases *parallel-reduction-fstE'*: $(Fst\ P) \gg X$

lemma *parallel-reduction-fstE*:

assumes $(Fst\ P) \gg X$
shows $(\exists P'.\ (P \gg P') \wedge X = (Fst\ P')) \vee (\exists A\ A'\ B.\ (P = Pair\ A\ B) \wedge (A \gg A') \wedge X = A')$
 $\langle proof \rangle$

inductive-cases *parallel-reduction-sndE'*: $(Snd\ P) \gg X$

lemma *parallel-reduction-sndE*:

assumes $(Snd\ P) \gg X$
shows $(\exists P'.\ (P \gg P') \wedge X = (Snd\ P')) \vee (\exists A\ B\ B'.\ (P = Pair\ A\ B) \wedge (B \gg B') \wedge X = B')$
 $\langle proof \rangle$

lemma *parallel-reduction-subst-inner*:

assumes $M \gg M'$
shows $X[z ::= M] \gg (X[z ::= M'])$
 $\langle proof \rangle$

lemma *parallel-reduction-subst*:

assumes $X \gg X'\ M \gg M'$
shows $X[z ::= M] \gg (X'[z ::= M'])$
 $\langle proof \rangle$

inductive *complete-development* :: 'a trm \Rightarrow 'a trm \Rightarrow bool (- >>> -) **where**

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

lemma *not-fn-prm*:

assumes $\bigwedge x T M. A \neq Fn\ x\ T\ M$
shows $\pi \cdot A \neq Fn\ x\ T\ M$
 $\langle proof \rangle$

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

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

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

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

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

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

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

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

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

lemma *parallel-reduces-diamond'*:
assumes $A >>* C\ A >> B$

shows $\exists D. (B \gg^* D) \wedge (C \gg D)$
<proof>

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

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

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

lemma *parallel-beta-reduces-equivalent:*
shows $(A \gg^* B) = (A \rightarrow\beta^* B)$
<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