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

Michael Rawson

May 26, 2024

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* $(s \mathbin{\#} ss)$ 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**
  $\pi =p \sigma \equiv \forall x.$ *preprm-apply* π x = *preprm-apply* σ x

definition *preprm-inv* :: $'a$ *preprm* $\Rightarrow$ $'a$ *preprm* **where**
  preprm-inv $\pi \equiv$ *rev* π

lemma *swp-apply-unequal*:
  assumes $x \neq y$
  shows *swp-apply* s $x \neq$ *swp-apply* s y
$\langle proof \rangle$

lemma *preprm-ext-reflexive*:
  shows $x =p x$
$\langle proof \rangle$

corollary *preprm-ext-reflp*:
  shows *reflp preprm-ext*
$\langle proof \rangle$

lemma *preprm-ext-symmetric*:
  assumes $x =p y$
  shows $y =p x$
$\langle proof \rangle$

corollary *preprm-ext-symp*:
  shows *symp preprm-ext*
$\langle proof \rangle$

lemma *preprm-ext-transitive*:
  assumes $x =p y$ **and** $y =p z$
  shows $x =p z$
$\langle proof \rangle$

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 ≠ 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 $\{x.$

$$\begin{aligned} preprm\text{-}apply\ (preprm\text{-}compose\ (preprm\text{-}unit\ a\ b)\ (preprm\text{-}unit\ b\ c))\ x \neq \\ preprm\text{-}apply\ (preprm\text{-}unit\ a\ c)\ x \\ \} = \{a,\ b\} \end{aligned}$$

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *preprm-ext-compose-left*:
  assumes $P =p S$
  shows *preprm-compose* π $P =p$ *preprm-compose* π S
$\langle proof \rangle$

lemma *preprm-ext-compose-right*:
  assumes $P =p S$
  shows *preprm-compose* P $\pi =p$ *preprm-compose* S π
$\langle proof \rangle$

lemma *preprm-ext-uncompose*:
  assumes $\pi =p \sigma$ *preprm-compose* π $P =p$ *preprm-compose* σ S
  shows $P =p S$
$\langle proof \rangle$

lemma *preprm-inv-compose*:
  shows *preprm-compose (preprm-inv* π*)* $\pi =p$ *preprm-id*
$\langle proof \rangle$

lemma *preprm-inv-involution*:
  shows *preprm-inv (preprm-inv* π*)* $= \pi$
$\langle proof \rangle$

lemma *preprm-inv-ext*:
  assumes $\pi =p \sigma$
  shows *preprm-inv* $\pi =p$ *preprm-inv* σ
$\langle proof \rangle$

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

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

lift-definition *prm-apply* :: $'a\ prm \Rightarrow {'a} \Rightarrow {'a}$ (**infix** $\$$ *140*) **is** *preprm-apply*
$\langle proof \rangle$

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

4

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)
$\langle proof \rangle$

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

interpretation $'a$ *prm*: *semigroup prm-compose*
$\langle proof \rangle$

interpretation $'a$ *prm*: *group prm-compose prm-id prm-inv*
$\langle proof \rangle$

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 {$} ($\sigma$ {$} *S*) = ($\pi$ $\diamond$ σ) {$} *S*
$\langle proof \rangle$

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

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

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

lemma *prm-set-id*:
  shows ε {$} *S* = *S*
$\langle proof \rangle$

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

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

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

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

lemma *ptrm-fvs-finite*:
  shows *finite* (*ptrm-fvs* X)
$\langle proof \rangle$

lemma *ptrm-prm-fvs*:
  shows *ptrm-fvs* ($\pi \cdot X$) $= \pi$ {\$} *ptrm-fvs* X
$\langle proof \rangle$

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

lemma *ptrm-alpha-equiv-prm*:
  assumes $X \approx Y$
  shows $\pi \cdot X \approx \pi \cdot Y$
$\langle proof \rangle$

lemma *ptrm-swp-transfer*:
  shows $[a \leftrightarrow b] \cdot X \approx Y \longleftrightarrow X \approx [a \leftrightarrow b] \cdot Y$
$\langle proof \rangle$

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
$\langle proof \rangle$

lemma *ptrm-prm-agreement-equiv*:
  assumes $\bigwedge a.\ a \in ds\ \pi\ \sigma \Longrightarrow a \notin$ *ptrm-fvs* M
  shows $\pi \cdot M \approx \sigma \cdot M$
$\langle proof \rangle$

lemma *ptrm-prm-unit-inaction*:
  assumes $a \notin$ *ptrm-fvs* $X\ b \notin$ *ptrm-fvs* X
  shows $[a \leftrightarrow b] \cdot X \approx X$
$\langle proof \rangle$

lemma *ptrm-alpha-equiv-reflexive*:
  shows $M \approx M$
$\langle proof \rangle$

corollary *ptrm-alpha-equiv-reflp*:
  shows *reflp ptrm-alpha-equiv*
$\langle proof \rangle$

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* $= {}'a \rightharpoonup type$

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 σ $\Rightarrow$ *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* $(\Gamma(a \mapsto S, b \mapsto T))$ $([a \leftrightarrow b] \cdot X)$

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *ptrm-infer-type-alpha-equiv*:

  assumes $X \approx Y$

  shows *ptrm-infer-type* Γ $X = $ *ptrm-infer-type* Γ Y

$\langle proof \rangle$

end
theory *SimplyTyped*
imports *PreSimplyTyped*
begin

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

lift-definition *Unit* :: $'a$ *trm* **is** *PUnit*$\langle proof \rangle$
lift-definition *Var* :: $'a \Rightarrow 'a$ *trm* **is** *PVar*$\langle proof \rangle$
lift-definition *App* :: $'a$ *trm* $\Rightarrow 'a$ *trm* $\Rightarrow 'a$ *trm* **is** *PApp* $\langle proof \rangle$
lift-definition *Fn* :: $'a \Rightarrow$ *type* $\Rightarrow 'a$ *trm* $\Rightarrow 'a$ *trm* **is** *PFn* $\langle proof \rangle$
lift-definition *Pair* :: $'a$ *trm* $\Rightarrow 'a$ *trm* $\Rightarrow 'a$ *trm* **is** *PPair* $\langle proof \rangle$
lift-definition *Fst* :: $'a$ *trm* $\Rightarrow 'a$ *trm* **is** *PFst* $\langle proof \rangle$
lift-definition *Snd* :: $'a$ *trm* $\Rightarrow 'a$ *trm* **is** *PSnd* $\langle proof \rangle$
lift-definition *fvs* :: $'a$ *trm* $\Rightarrow 'a$ *set* **is** *ptrm-fvs* $\langle proof \rangle$
lift-definition *prm* :: $'a$ *prm* $\Rightarrow 'a$ *trm* $\Rightarrow 'a$ *trm* (**infixr** $\cdot$ *150*) **is** *ptrm-apply-prm*
  $\langle proof \rangle$
lift-definition *depth* :: $'a$ *trm* $\Rightarrow$ *nat* **is** *size* $\langle proof \rangle$

lemma *depth-prm*:

  shows *depth* $(\pi \cdot A) = $ *depth* A

$\langle proof \rangle$

lemma *depth-app*:

  shows *depth* $A <$ *depth* $(App\ A\ B)$ *depth* $B <$ *depth* $(App\ A\ B)$

$\langle proof \rangle$

lemma *depth-fn*:

  shows *depth* $A <$ *depth* $(Fn\ x\ T\ A)$

$\langle proof \rangle$

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*

$App\ A\ B = App\ C\ D \implies A = C$
$App\ A\ B = App\ C\ D \implies B = D$
$Fn\ x\ T\ A = Fn\ y\ S\ B \implies$
$\quad (x = y \land T = S \land A = B) \lor (x \neq y \land T = S \land x \notin fvs\ B \land A = [x \leftrightarrow y] \cdot$
$B)$
$Pair\ A\ B = Pair\ C\ D \implies A = C$
$Pair\ A\ B = Pair\ C\ D \implies B = D$
$Fst\ P = Fst\ Q \implies P = Q$
$Snd\ P = Snd\ Q \implies P = Q$
$\langle proof \rangle$

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
   $\pi \cdot Unit = Unit$
   $\pi \cdot Var\ x = Var\ (\pi\ \$\ x)$
   $\pi \cdot App\ A\ B = App\ (\pi \cdot A)\ (\pi \cdot B)$
   $\pi \cdot Fn\ x\ T\ A = Fn\ (\pi\ \$\ x)\ T\ (\pi \cdot A)$
   $\pi \cdot Pair\ A\ B = Pair\ (\pi \cdot A)\ (\pi \cdot B)$
   $\pi \cdot Fst\ P = Fst\ (\pi \cdot P)$
   $\pi \cdot Snd\ P = Snd\ (\pi \cdot P)$
  $\langle proof \rangle$

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
   $fvs\ Unit = \{\}$ **and**
   $fvs\ (Var\ x) = \{x\}$
   $fvs\ (App\ A\ B) = fvs\ A \cup fvs\ B$
   $fvs\ (Fn\ x\ T\ A) = fvs\ A - \{x\}$
   $fvs\ (Pair\ A\ B) = fvs\ A \cup fvs\ B$
   $fvs\ (Fst\ P) = fvs\ P$
   $fvs\ (Snd\ P) = fvs\ P$
$\langle proof \rangle$

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 $[a \leftrightarrow b] \cdot Var\ x = Var\ x$
$\langle proof \rangle$

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

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

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

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

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

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*
$\langle proof \rangle$

context *fresh* **begin**

lemma *fresh-fn*:
  fixes $x :: 'a$ **and** $S :: 'a\ set$
  assumes *finite S*
  shows $\exists y\ B.\ y \notin S \wedge B = [y \leftrightarrow x] \cdot A \wedge (Fn\ x\ T\ A = Fn\ y\ T\ B)$
$\langle proof \rangle$

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*
$\langle proof \rangle$

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

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

16

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

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

inductive *typing* :: $'a\ typing\text{-}ctx \Rightarrow 'a\ trm \Rightarrow type \Rightarrow bool$ $(\text{-} \vdash \text{-} : \text{-})$ **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 $\Delta \vdash \pi \cdot M : \tau$
$\langle proof \rangle$

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

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

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

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

lemma *typing-fnE*:

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

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

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

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

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

lift-definition *infer* :: $'a \textit{ typing-ctx} \Rightarrow {}'a \textit{ trm} \Rightarrow \textit{type option}$ **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') $\Rightarrow$ (*if* $\tau = \tau'$ *then Some* σ *else None*)
    | - $\Rightarrow$ *None*
)
    infer Γ (*Fn x* τ *A*) = (*case infer* ($\Gamma(x \mapsto \tau)$) *A of*
      Some σ $\Rightarrow$ *Some* (*TArr* τ σ)
    | *None* $\Rightarrow$ *None*
)
    infer Γ (*Pair A B*) = (*case* (*infer* Γ *A*, *infer* Γ *B*) *of*
      (*Some* τ, *Some* σ) $\Rightarrow$ *Some* (*TPair* τ σ)
    | - $\Rightarrow$ *None*
)
    infer Γ (*Fst P*) = (*case infer* Γ *P of*
      (*Some* (*TPair* τ σ)) $\Rightarrow$ *Some* τ
    | - $\Rightarrow$ *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 $\tau = TUnit$
⟨*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 $\exists \sigma.\ infer\ \Gamma\ A = Some\ (TArr\ \sigma\ \tau) \land infer\ \Gamma\ B = Some\ \sigma$
⟨*proof*⟩

lemma *infer-fnE*:
  assumes *infer Γ (Fn x T A) = Some τ*
  shows $\exists \sigma.\ \tau = TArr\ T\ \sigma \land infer\ (\Gamma(x \mapsto T))\ A = Some\ \sigma$
⟨*proof*⟩

lemma *infer-pairE*:
  assumes *infer Γ (Pair A B) = Some τ*
  shows $\exists\ T\ S.\ \tau = TPair\ T\ S \land infer\ \Gamma\ A = Some\ T \land infer\ \Gamma\ B = Some\ S$
⟨*proof*⟩

lemma *infer-fstE*:
  assumes *infer Γ (Fst P) = Some τ*
  shows $\exists\ T\ S.\ infer\ \Gamma\ P = Some\ (TPair\ T\ S) \land \tau = T$
⟨*proof*⟩

lemma *infer-sndE*:
  assumes *infer Γ (Snd P) = Some τ*
  shows $\exists\ T\ S.\ infer\ \Gamma\ P = Some\ (TPair\ T\ S) \land \tau = S$
⟨*proof*⟩

lemma *infer-sound*:
  assumes *infer Γ M = Some τ*
  shows $\Gamma \vdash M : \tau$
⟨*proof*⟩

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

19
```

**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*

20

**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** $(Var\ x)[y ::= M] = Var\ x$
⟨*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* $\Rightarrow$ $'a$ *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*⟩

23

**lemma** *beta-reduction-left-pairE*:
  **assumes** $(Pair\ A\ B) \to \beta\ M'$
  **shows** $(\exists\ A'.\ (A \to \beta\ A') \land M' = (Pair\ A'\ B)) \lor (\exists\ B'.\ (B \to \beta\ B') \land M' = (Pair$
$A\ B'))$
$\langle proof \rangle$

**lemma** *beta-reduction-left-fstE*:
  **assumes** $(Fst\ P) \to \beta\ M'$
  **shows** $(\exists\ P'.\ (P \to \beta\ P') \land M' = (Fst\ P')) \lor (\exists\ A\ B.\ P = (Pair\ A\ B) \land M' =$
$A)$
$\langle proof \rangle$

**lemma** *beta-reduction-left-sndE*:
  **assumes** $(Snd\ P) \to \beta\ M'$
  **shows** $(\exists\ P'.\ (P \to \beta\ P') \land M' = (Snd\ P')) \lor (\exists\ A\ B.\ P = Pair\ A\ B \land M' = B)$
$\langle proof \rangle$

**lemma** *preservation′*:
  **assumes** $\Gamma \vdash M : \tau$ **and** $M \to \beta\ M'$
  **shows** $\Gamma \vdash M' : \tau$
$\langle proof \rangle$

**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** $NF\ M \lor (\exists\ M'.\ (M \to \beta\ M'))$
$\langle proof \rangle$

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

**lemma** *beta-reduces-composition*:
  **assumes** $A' \to \beta^*\ A''$ **and** $A \to \beta^*\ A'$
  **shows** $A \to \beta^*\ A''$
$\langle proof \rangle$

**lemma** *beta-reduces-fvs*:
  **assumes** $A \to \beta^*\ A'$
  **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** $NF\ M' \lor (\exists\ M''.\ (M' \rightarrow \beta\ M''))$
*⟨proof⟩*

**inductive** *parallel-reduction* :: $'a\ trm \Rightarrow\ 'a\ trm \Rightarrow bool$ (- >> -) **where**
  *refl*: $A >> A$

| *beta*: ⟦*A* >> *A′*; *B* >> *B′*⟧ ⟹ (*App* (*Fn x T A*) *B*) >> (*A′*[*x* ::= *B′*])
| *eta*: *A* >> *A′* ⟹ (*Fn x T A*) >> (*Fn x T A′*)
| *app*: ⟦*A* >> *A′*; *B* >> *B′*⟧ ⟹ (*App A B*) >> (*App A′ B′*)
| *pair*: ⟦*A* >> *A′*; *B* >> *B′*⟧ ⟹ (*Pair A B*) >> (*Pair A′ B′*)
| *fst1*: *P* >> *P′* ⟹ (*Fst P*) >> (*Fst P′*)
| *fst2*: *A* >> *A′* ⟹ (*Fst* (*Pair A B*)) >> *A′*
| *snd1*: *P* >> *P′* ⟹ (*Snd P*) >> (*Snd P′*)
| *snd2*: *B* >> *B′* ⟹ (*Snd* (*Pair A B*)) >> *B′*

**lemma** *parallel-reduction-prm*:
  **assumes** *A* >> *A′*
  **shows** (π · *A*) >> (π · *A′*)
⟨*proof*⟩

**lemma** *parallel-reduction-fvs*:
  **assumes** *A* >> *A′*
  **shows** *fvs A′* ⊆ *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* ∨ (∃ *A′*. (*A* >> *A′*) ∧ *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*) ∨
    (∃ *A′ B′*. (*A* >> *A′*) ∧ (*B* >> *B′*) ∧ *X* = (*A′*[*x* ::= *B′*])) ∨
    (∃ *A′ B′*. ((*Fn x T A*) >> (*Fn x T A′*)) ∧ (*B* >> *B′*) ∧ *X* = (*App* (*Fn x T A′*) *B′*))

⟨*proof*⟩

**inductive-cases** *parallel-reduction-nonredexE′*: (*App A B*) >> *X*

26

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

**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** $A \ggg A'$
  **shows** $(\pi \cdot A) \ggg (\pi \cdot A')$
⟨*proof*⟩

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

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

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

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

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

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

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

**lemma** *parallel-reduces-diamond′*:
  **assumes** $A \gg^* C\ A \gg B$

**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** $A \to\beta\ B$
  **shows** $A >> B$
⟨*proof*⟩

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

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

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

**end**
**end**