

C++.T Formalization in Isar

Jeremy Siek and Walid Taha

December 16, 2005

Contents

theory *Cpp* = *Main*:

types *tmname* = *nat*

types *field* = *nat*

datatype *ty*

= *TVar nat*

| *TMem ty field*

| *TmId tmname ty ty*

| *TArrow ty ty*

| *TInt*

consts *FTV* :: *ty* \Rightarrow *nat set*

primrec

FTV (TVar i) = {*i*}

FTV (TMem τ a) = *FTV τ*

FTV (TmId t τ 1 τ 2) = *FTV τ 1* \cup *FTV τ 2*

FTV (TArrow τ 1 τ 2) = *FTV τ 1* \cup *FTV τ 2*

FTV TInt = {}

consts

type-value :: *ty set*

inductive *type-value intros*

VVar[intro!]: *TVar i* \in *type-value*

VInt[intro!]: *TInt* \in *type-value*

VMem[intro!]: $\llbracket \tau \in \text{type-value}; \text{FTV } \tau \neq \{\} \rrbracket$

\implies *TMem τ a* \in *type-value*

VTmId[intro!]: $\llbracket \tau$ 1 \in *type-value*; τ 2 \in *type-value* \rrbracket

\implies *TmId t τ 1 τ 2* \in *type-value*

VArrow[intro!]: $\llbracket \tau$ 1 \in *type-value*; τ 2 \in *type-value* \rrbracket

\implies *TArrow τ 1 τ 2* \in *type-value*

inductive-cases *ty-val-inv[elim!]*:

TMem τ a \in *type-value*

TmId t τ 1 τ 2 \in *type-value*

TArrow τ 1 τ 2 \in *type-value*

consts

residual-type :: *ty set*

inductive *residual-type intros*

RInt[intro!]: *TInt* \in *residual-type*

RTmId[intro!]: $\llbracket \tau$ 1 \in *residual-type*; τ 2 \in *residual-type* \rrbracket

$\implies TmId\ t\ \tau1\ \tau2 \in residual\text{-}type$
 $RArrow[intro!]: \llbracket \tau1 \in residual\text{-}type; \tau2 \in residual\text{-}type \rrbracket$
 $\implies TArrow\ \tau1\ \tau2 \in residual\text{-}type$

lemma *closed-value-residual*:

$\llbracket \tau \in type\text{-}value; FTV\ \tau = \{\} \rrbracket \implies \tau \in residual\text{-}type$
by (*induct rule*: *type-value.induct, auto*)

lemma *residual-value*: $\tau \in residual\text{-}type \implies \tau \in type\text{-}value$

by (*induct rule*: *residual-type.induct, auto*)

lemma *residual-closed*: $\tau \in residual\text{-}type \implies FTV\ \tau = \{\}$

by (*induct rule*: *residual-type.induct, auto*)

types *tm-used* = (*tmname* \times *ty* \times *ty*) *set*

types *tyvars* = *nat set*

datatype *memkind* = *Fun* | *Type*

types *member* = *memkind* \times *field* \times *ty*

types *tm-set* = (*ty* \times *member*) *set*

constdefs *tms* :: *tm-set* \Rightarrow *tmname set*

tms *T* $\equiv \{t. \exists\ \tau1\ \tau2\ m. (TmId\ t\ \tau1\ \tau2, m) \in T\}$

consts *wf-ty* :: (*tm-set* \times *tyvars* \times *ty*) *set*

syntax (*xsymbol*)

wf-ty :: *tm-set* \Rightarrow *tyvars* \Rightarrow *ty* \Rightarrow *bool* (- | - \vdash - *wf* [52,52,52] 51)

translations

$T \mid V \vdash \tau\ wf \iff (T, V, \tau) \in wf\text{-}ty$

inductive *wf-ty intros*

WFTVar[*intro!*]: $i \in V \implies T \mid V \vdash TVar\ i\ wf$

WFTMem[*intro!*]: $T \mid V \vdash \tau\ wf \implies T \mid V \vdash TMem\ \tau\ a\ wf$

WFTmId[*intro!*]: $\llbracket t \in tms\ T;$

$T \mid V \vdash \tau1\ wf; T \mid V \vdash \tau2\ wf \rrbracket$

$\implies T \mid V \vdash (TmId\ t\ \tau1\ \tau2)\ wf$

WFTArrow[*intro!*]: $\llbracket T \mid V \vdash \tau1\ wf; T \mid V \vdash \tau2\ wf \rrbracket$

$\implies T \mid V \vdash (TArrow\ \tau1\ \tau2)\ wf$

WFTInt[*intro!*]: $T \mid V \vdash TInt\ wf$

inductive-cases *wf-ty-inv*[*elim!*]:

$T \mid V \vdash (TVar\ i)\ wf$

$T \mid V \vdash (TMem\ \tau\ a)\ wf$

$T \mid V \vdash (TmId\ t\ \tau1\ \tau2)\ wf$

$T \mid V \vdash (TArrow\ \tau1\ \tau2)\ wf$

consts *lookup* :: *tm-set* \Rightarrow *tm-set*

consts *ty-eval* :: (*tm-set* \times *tyvars* \times *ty* \times *ty option*) *set*

syntax (*xsymbol*)

ty-eval :: *tm-set* \Rightarrow *tyvars* \Rightarrow *ty* \Rightarrow *ty option* \Rightarrow *bool* (- | - \vdash - \Downarrow - [52,52,52,52] 51)

translations

$T \mid V \vdash \tau1\ \Downarrow\ \tau2 \iff (T, V, \tau1, \tau2) \in ty\text{-}eval$

inductive *ty-eval intros*

CVarT[*intro!*]: $i \in V \implies T \mid V \vdash TVar\ i\ \Downarrow\ Some\ (TVar\ i)$

CMemT1[*intro!*]: $\llbracket T \mid V \vdash \tau\ \Downarrow\ Some\ (TmId\ t\ \tau1\ \tau2);$

$$\begin{aligned}
& FTV \tau 1 = \{\}; FTV \tau 2 = \{\}; \\
& (TmId \ t \ \tau 1 \ \tau 2, \ Type, \ a, \ \tau') \in \text{lookup } T; \\
& T \mid \{\} \vdash \tau' \Downarrow \text{Some } \tau'' \\
& \implies T \mid V \vdash TMem \ \tau \ a \Downarrow \text{Some } \tau'' \\
CMemT2[intro!]: & \llbracket T \mid V \vdash \tau \Downarrow \text{Some } \tau'; FTV \ \tau' \neq \{\} \rrbracket \\
& \implies T \mid V \vdash TMem \ \tau \ a \Downarrow \text{Some } (TMem \ \tau' \ a) \\
CTmT[intro!]: & \llbracket t \in \text{tms } T; \\
& T \mid V \vdash \tau 1 \Downarrow \text{Some } \tau 1'; \\
& T \mid V \vdash \tau 2 \Downarrow \text{Some } \tau 2' \rrbracket \\
& \implies T \mid V \vdash (TmId \ t \ \tau 1 \ \tau 2) \Downarrow \text{Some } (TmId \ t \ \tau 1' \ \tau 2') \\
CArrowT[intro!]: & \llbracket T \mid V \vdash \tau 1 \Downarrow \text{Some } \tau 1'; \\
& T \mid V \vdash \tau 2 \Downarrow \text{Some } \tau 2' \rrbracket \\
& \implies T \mid V \vdash (TArrow \ \tau 1 \ \tau 2) \Downarrow \text{Some } (TArrow \ \tau 1' \ \tau 2') \\
CIntT[intro!]: & T \mid V \vdash TInt \Downarrow \text{Some } TInt \\
\\
CVarTE[intro!]: & i \notin V \implies T \mid V \vdash TVar \ i \Downarrow \text{None} \\
CMemTE1[intro!]: & \llbracket T \mid V \vdash \tau \Downarrow \text{Some } (TmId \ t \ \tau 1 \ \tau 2); \\
& FTV \ \tau 1 = \{\}; FTV \ \tau 2 = \{\}; \\
& \forall \tau. (TmId \ t \ \tau 1 \ \tau 2, \ Type, \ a, \ \tau) \notin \text{lookup } T \rrbracket \\
& \implies T \mid V \vdash (TMem \ \tau \ a) \Downarrow \text{None} \\
CMemTE2[intro!]: & \llbracket T \mid V \vdash \tau \Downarrow \text{Some } \tau'; FTV \ \tau' = \{\}; \\
& \neg(\exists t \ \tau 1 \ \tau 2. \ \tau' \neq TmId \ t \ \tau 1 \ \tau 2) \rrbracket \\
& \implies T \mid V \vdash (TMem \ \tau \ a) \Downarrow \text{None} \\
CMemTE3[intro!]: & \llbracket T \mid V \vdash \tau \Downarrow \text{None} \rrbracket \\
& \implies T \mid V \vdash (TMem \ \tau \ a) \Downarrow \text{None} \\
CTmTE1[intro!]: & \llbracket t \notin \text{tms } T \rrbracket \\
& \implies T \mid V \vdash (TmId \ t \ \tau 1 \ \tau 2) \Downarrow \text{None} \\
CTmTE2[intro!]: & \llbracket T \mid V \vdash \tau 1 \Downarrow a1; T \mid V \vdash \tau 2 \Downarrow a2; \\
& a1 = \text{None} \vee a2 = \text{None} \rrbracket \\
& \implies T \mid V \vdash (TmId \ t \ \tau 1 \ \tau 2) \Downarrow \text{None} \\
CArrowTE[intro!]: & \llbracket T \mid V \vdash \tau 1 \Downarrow a1; T \mid V \vdash \tau 2 \Downarrow a2; \\
& a1 = \text{None} \vee a2 = \text{None} \rrbracket \\
& \implies T \mid V \vdash (TArrow \ \tau 1 \ \tau 2) \Downarrow \text{None}
\end{aligned}$$

inductive-cases *ty-eval-inv*[*elim!*]:

$$\begin{aligned}
& T \mid V \vdash (TVar \ i) \Downarrow a \\
& T \mid V \vdash TInt \Downarrow a \\
& T \mid V \vdash (TArrow \ \tau 1 \ \tau 2) \Downarrow a \\
& T \mid V \vdash (TMem \ \tau \ f) \Downarrow a \\
& T \mid V \vdash (TmId \ t \ \tau 1 \ \tau 2) \Downarrow a
\end{aligned}$$

lemma *eval-type-value*: $T \mid V \vdash \tau \Downarrow \text{Some } \tau' \implies \tau' \in \text{type-value}$

proof –

$$\begin{aligned}
& \{ \text{fix } T \ V \ \tau \ a \\
& \quad \text{have } T \mid V \vdash \tau \Downarrow a \implies (\bigwedge \tau'. a = \text{Some } \tau' \implies \tau' \in \text{type-value}) \\
& \quad \text{by (induct rule: ty-eval.induct, auto)} \\
& \} \text{ thus } T \mid V \vdash \tau \Downarrow \text{Some } \tau' \implies \tau' \in \text{type-value} \text{ by simp}
\end{aligned}$$

qed

lemma *eval-type-residual*:

$$\llbracket T \mid V \vdash \tau \Downarrow \text{Some } \tau'; V = \{\} \rrbracket \implies \tau' \in \text{residual-type}$$

proof –

$$\begin{aligned}
& \{ \text{fix } T \ V \ \tau \ a \\
& \quad \text{have } \llbracket T \mid V \vdash \tau \Downarrow a \rrbracket \\
& \quad \implies (\bigwedge \tau'. \llbracket a = \text{Some } \tau'; V = \{\} \rrbracket) \\
& \quad \implies \tau' \in \text{residual-type}
\end{aligned}$$

```

    apply (induct rule: ty-eval.induct)
    using residual-closed apply auto
    done
  } thus [ [ T | V ⊢ τ ↓ Some τ'; V = {} ] ]
    ⇒ τ' ∈ residual-type by simp
qed

```

lemma *value-idempotent*:

```

[ [ T | V ⊢ τ ↓ a; T | V ⊢ τ wf; τ ∈ type-value ] ] ⇒ a = Some τ
by (induct rule: ty-eval.induct, auto)

```

datatype *expr*

```

= EVar
| EInt int
| EObj ty
| EMem ty field
| EApp expr expr

```

consts

residual-expr :: *expr* set

inductive *residual-expr* **intros**

```

RsEVar[intro!]: EVar ∈ residual-expr
RsEInt[intro!]: EInt n ∈ residual-expr
RsEObj[intro!]: τ ∈ residual-type ⇒ EObj τ ∈ residual-expr
RsEMem[intro!]: τ ∈ residual-type ⇒ EMem τ f ∈ residual-expr
RsEApp[intro!]: [ [ e1 ∈ residual-expr; e2 ∈ residual-expr ] ] ⇒ EApp e1 e2 ∈ residual-expr

```

inductive-cases *residual-expr-inv* [elim!]:

(*EApp* *e1* *e2*) ∈ *residual-expr*

Compile expressions by evaluating nested type expressions

types *fun-used* = (*ty* × *nat*) set

consts *comp-expr* :: (*tm-set* × *tyvars* × *expr* × *expr*) set

syntax (*xsymbol*)

comp-expr :: *tm-set* ⇒ *tyvars* ⇒ *expr* ⇒ *expr* ⇒ *bool* (- | - ⊢ - ↓ - [52,52,52,52] 51)

translations

$T \mid V \vdash e \downarrow e' == (T, V, e :: \text{expr}, e') \in \text{comp-expr}$

inductive *comp-expr* **intros**

```

CEVar: T | V ⊢ EVar ↓ EVar
CEInt: T | V ⊢ EInt n ↓ (EInt n)
CEObj: T | V ⊢ τ ↓ Some τ'
    ⇒ T | V ⊢ EObj τ ↓ (EObj τ')
CEMem: [ [ T | V ⊢ τ ↓ Some τ' ] ]
    ⇒ T | V ⊢ EMem τ f ↓ (EMem τ' f)
CEApp: [ [ T | V ⊢ e1 ↓ e1'; T | V ⊢ e2 ↓ e2' ] ]
    ⇒ T | V ⊢ EApp e1 e2 ↓ (EApp e1' e2')

```

consts *subexpr* :: (*expr* × *expr*) set

inductive *subexpr* **intros**

```

SubRefl[intro!]: (e, e) ∈ subexpr
SubApp[intro!]: (e, e1) ∈ subexpr ∨ (e, e2) ∈ subexpr ⇒ (e, EApp e1 e2) ∈ subexpr

```

inductive-cases *subexpr-app-inv*[elim!]:

```

(e, EVar) ∈ subexpr
(e, EInt n) ∈ subexpr

```

$(e, EObj \tau) \in subexpr$
 $(e, EMem \tau f) \in subexpr$
 $(e, EApp e1 e2) \in subexpr$

constdefs

$ty-used-in :: expr \Rightarrow ty \ set$
 $ty-used-in \ e \equiv \{ \tau. (EObj \ \tau, \ e) \in subexpr \vee (\exists f. (EMem \ \tau \ f, \ e) \in subexpr) \}$

declare $ty-used-in-def[simp]$

consts $ty-eq :: (tm-set \times ty \times ty) \ set$

syntax ($xsymbol$)

$ty-eq :: tm-set \Rightarrow ty \Rightarrow ty \Rightarrow bool \quad (- \vdash - \equiv - \ [52,52,52] \ 51)$

translations

$T \vdash \tau1 \equiv \tau2 == (T :: tm-set, \tau1, \tau2) \in ty-eq$

inductive $ty-eq$ **intros**

$Mem: (\tau, Type, a, \tau') \in lookup \ T \Longrightarrow T \vdash TMem \ \tau \ a \equiv \tau'$
 $MemC[intro!]: T \vdash \tau \equiv \tau' \Longrightarrow T \vdash TMem \ \tau \ a \equiv TMem \ \tau' \ a$
 $TmC[intro!]: \llbracket T \vdash \tau1 \equiv \tau1'; T \vdash \tau2 \equiv \tau2' \rrbracket$
 $\Longrightarrow T \vdash TmId \ t \ \tau1 \ \tau2 \equiv TmId \ t \ \tau1' \ \tau2'$
 $ArrowC[intro!]: \llbracket T \vdash \tau1 \equiv \tau1'; T \vdash \tau2 \equiv \tau2' \rrbracket$
 $\Longrightarrow T \vdash TArrow \ \tau1 \ \tau2 \equiv TArrow \ \tau1' \ \tau2'$
 $Refl[intro!]: T \vdash \tau \equiv \tau$
 $Symm[sym]: T \vdash \tau1 \equiv \tau2 \Longrightarrow T \vdash \tau2 \equiv \tau1$
 $Trans[trans]: \llbracket T \vdash \tau1 \equiv \tau2; T \vdash \tau2 \equiv \tau3 \rrbracket \Longrightarrow T \vdash \tau1 \equiv \tau3$

lemma $ty-eval-sound$:

$T \mid V \vdash \tau \Downarrow a \Longrightarrow (\bigwedge \tau'. a = Some \ \tau' \Longrightarrow T \vdash \tau \equiv \tau')$

proof (*induct rule: ty-eval.induct*)

fix $T \ V \ i \ \tau'$ — Case $CVarT$
assume $Some \ (TVar \ i) = Some \ \tau'$ **with** $Refl$
show $T \vdash (TVar \ i) \equiv \tau'$ **by** $auto$

next

fix $T \ V \ \tau \ \tau' \ \tau'' \ \tau1 \ \tau2 \ a \ t \ \tau'a$ — Case $CMemT1$
assume $IH1: \bigwedge \tau'. Some \ (TmId \ t \ \tau1 \ \tau2) = Some \ \tau' \Longrightarrow (T, \ \tau, \ \tau') \in ty-eq$
and $TM: (TmId \ t \ \tau1 \ \tau2, \ Type, \ a, \ \tau') \in lookup \ T$
and $IH2: \bigwedge \tau'a. Some \ \tau'' = Some \ \tau'a \Longrightarrow (T, \ \tau', \ \tau'a) \in ty-eq$
and $TPP: Some \ \tau'' = Some \ \tau'a$
from $IH1$ **have** $TTM: T \vdash \tau \equiv TmId \ t \ \tau1 \ \tau2$ **by** $simp$
from $IH2$ TPP **have** $T \vdash \tau' \equiv \tau'a$ **by** $auto$
hence $TPA: T \vdash \tau' \equiv \tau'a$ **by** $simp$
from TM **have** $TMTP: T \vdash TMem \ (TmId \ t \ \tau1 \ \tau2) \ a \equiv \tau'$ **by** (*rule Mem*)
from TTM **have** $A: T \vdash TMem \ \tau \ a \equiv TMem \ (TmId \ t \ \tau1 \ \tau2) \ a$ **by** (*rule MemC*)
from A $TMTP$ **have** $B: T \vdash TMem \ \tau \ a \equiv \tau'$ **by** (*rule Trans*)
from B TPA **show** $T \vdash TMem \ \tau \ a \equiv \tau'a$ **by** (*rule Trans*)

next

fix $T \ V \ \tau \ \tau' \ a \ \tau'a$ — Case $CMemT2$
assume $IH: \bigwedge \tau'a. Some \ \tau' = Some \ \tau'a \Longrightarrow (T, \ \tau, \ \tau'a) \in ty-eq$
and $TM: Some \ (TMem \ \tau' \ a) = Some \ \tau'a$
from IH **have** $T \vdash \tau \equiv \tau'$ **by** $simp$
hence $T \vdash TMem \ \tau \ a \equiv TMem \ \tau' \ a$ **by** (*rule MemC*)
with TM **show** $T \vdash TMem \ \tau \ a \equiv \tau'a$ **by** $simp$

next

fix $T \ V \ \tau1 \ \tau1' \ \tau2 \ \tau2' \ t \ \tau'$ — Case $CTmT$
assume $IH1: \bigwedge \tau'. Some \ \tau1' = Some \ \tau' \Longrightarrow (T, \ \tau1, \ \tau') \in ty-eq$
and $IH2: \bigwedge \tau'. Some \ \tau2' = Some \ \tau' \Longrightarrow (T, \ \tau2, \ \tau') \in ty-eq$

```

    and TM: Some (TmId t τ1' τ2') = Some τ'
  from IH1 have T ⊢ τ1 ≡ τ1' by simp
  also from IH2 have T ⊢ τ2 ≡ τ2' by simp
  ultimately have T ⊢ TmId t τ1 τ2 ≡ TmId t τ1' τ2' by (rule TmC)
  with TM show T ⊢ TmId t τ1 τ2 ≡ τ' by simp
next
fix T V τ1 τ1' τ2 τ2' τ'
assume IH1:  $\bigwedge \tau'. \text{Some } \tau1' = \text{Some } \tau' \implies (T, \tau1, \tau') \in \text{ty-eq}$ 
  and IH2:  $\bigwedge \tau'. \text{Some } \tau2' = \text{Some } \tau' \implies (T, \tau2, \tau') \in \text{ty-eq}$ 
  and A: Some (TArrow τ1' τ2') = Some τ'
from IH1 have IH1a: T ⊢ τ1 ≡ τ1' by simp
from IH2 have IH2a: T ⊢ τ2 ≡ τ2' by simp
from IH1a IH2a have B: T ⊢ TArrow τ1 τ2 ≡ TArrow τ1' τ2' by (rule ArrowC)
with A show T ⊢ TArrow τ1 τ2 ≡ τ' by simp
next
fix T V m τ' assume Some TInt = Some τ'
with Reft show T ⊢ TInt ≡ τ' by auto
next
fix T V i m τ' assume None = Some (τ'::ty) thus (T, TVar i, τ') ∈ ty-eq by auto
next
fix T V τ a τ' assume None = Some (τ'::ty) thus (T, TMem τ a, τ') ∈ ty-eq by auto
next
fix T V τ a τ' assume None = Some (τ'::ty) thus (T, TMem τ a, τ') ∈ ty-eq by auto
next
fix T V τ a τ' assume None = Some (τ'::ty) thus (T, TMem τ a, τ') ∈ ty-eq by auto
next
fix T V t τ1 τ2 τ' assume None = Some (τ'::ty) thus T ⊢ TmId t τ1 τ2 ≡ τ' by auto
next
fix T V τ1 τ2 τ' assume None = Some (τ'::ty) thus T ⊢ TArrow τ1 τ2 ≡ τ' by auto
next
fix T V τ1 τ2 a1 a2 t τ' assume None = Some (τ'::ty)
thus (T, TmId t τ1 τ2, τ') ∈ ty-eq by auto
qed

constdefs compty :: tm-set ⇒ ty set
  compty T ≡ { τ. ∃ m. (τ, m) ∈ T ∧ τ ∈ residual-type }
declare compty-def[simp]

consts wt-expr :: (tm-set × tyvars × ty option × expr × ty option) set
syntax (xsymbol)
  wt-expr :: tm-set ⇒ tyvars ⇒ ty option ⇒ expr ⇒ ty option ⇒ bool (- | - | - ⊢ - : - [52,52,52,52,52] 51)
translations
  T | V | τ1 ⊢ e : τ2 == (T, V, τ1, e::expr, τ2) ∈ wt-expr
inductive wt-expr intros
  TApp1[intro]:
    [[ T | V | τ ⊢ e1 : Some (TArrow τ1 τ2); T | V | τ ⊢ e2 : Some τ1 ]]
    ⇒ T | V | τ ⊢ EApp e1 e2 : Some τ2
  TApp2[intro]:
    [[ T | V | τ ⊢ e1 : a1; T | V | τ ⊢ e2 : a2; a1 = None ∨ a2 = None ]]
    ⇒ T | V | τ ⊢ EApp e1 e2 : None
  TVar[intro]: a = (if FTV τ = {} then Some τ else None)
    ⇒ T | V | Some τ ⊢ EVar : a
  TInt[intro]: T | V | τ ⊢ EInt n : Some (TInt)
  TObj[intro]: [[ T | V ⊢ TmId t τ1 τ2 wf;
    FTV τ1 = {}; FTV τ2 = {};
    TmId t τ1 τ2 ∈ compty T ]]

```

$\implies T \mid V \mid \tau \vdash EObj (TmId\ t\ \tau1\ \tau2) : Some\ (TmId\ t\ \tau1\ \tau2)$
 $TEObjE[intro]: FTV\ \tau1 \neq \{\} \implies T \mid V \mid \tau2 \vdash EObj\ \tau1 : None$
 $TEMem[intro]: \llbracket T \mid V \vdash \tau\ wf;$
 $\quad (\tau, Fun, f, TArrow\ \tau1\ \tau2) \in T \rrbracket$
 $\implies T \mid V \mid \tau3 \vdash EMem\ \tau\ f : Some\ (TArrow\ \tau1\ \tau2)$
 $TEMemE[intro]: FTV\ \tau \neq \{\} \implies T \mid V \mid \tau1 \vdash EMem\ \tau\ f : None$

inductive-cases *wt-funval-inv*:

$T \mid V \mid \tau3 \vdash v : Some\ (TArrow\ \tau1\ \tau2)$

inductive-cases *wt-mem-inv*:

$T \mid V \mid \tau2 \vdash (EMem\ \tau\ f) : Some\ (TArrow\ \tau1\ \tau')$

inductive-cases *wt-var-inv*:

$T \mid V \mid None \vdash EVar : Some\ \tau$

inductive-cases *wt-app-inv*:

$T \mid V \mid \tau' \vdash EApp\ e1\ e2 : Some\ \tau$

inductive-cases *wt-obj-inv*:

$T \mid V \mid \tau' \vdash EObj\ \tau : Some\ \tau''$

constdefs

closed-tm-set :: *tm-set* \Rightarrow *bool*

closed-tm-set $T \equiv (\forall\ \tau\ k\ f\ \tau'. (\tau, k, f, \tau') \in T \longrightarrow FTV\ \tau' = \{\})$

declare *closed-tm-set-def*[*simp*]

consts *subst-expr* :: *expr* \Rightarrow *expr* \Rightarrow *expr*

primrec

subst-expr *EVar* $e = e$

subst-expr (*EInt* n) $e = EInt\ n$

subst-expr (*EObj* τ) $e = EObj\ \tau$

subst-expr (*EMem* $\tau\ f$) $e = (EMem\ \tau\ f)$

subst-expr (*EApp* $e1\ e2$) $e = (EApp\ (subst-expr\ e1\ e)\ (subst-expr\ e2\ e))$

lemma *subst-expr-type*:

$T \mid V \mid Some\ \tau1 \vdash e : a \implies$

$(\bigwedge\ \tau2\ v. \llbracket a = Some\ \tau2; T \mid V \mid None \vdash v : Some\ \tau1 \rrbracket$

$\implies T \mid V \mid None \vdash subst-expr\ e\ v : Some\ \tau2)$

apply (*induct* rule: *wt-expr.induct*)

apply *auto*

apply *blast*

apply (*case-tac* $FTV\ \tau = \{\}$)

apply *auto*

apply (*rule* *TEObj*)

apply *auto*

done

types *fun-set* = $(ty \times nat \times (ty \times ty \times expr))\ set$

consts *run-expr* :: $(fun-set \times tm-set \times expr \times expr\ option)\ set$

syntax (*xsymbol*)

run-expr :: *fun-set* \Rightarrow *tm-set* \Rightarrow *expr* \Rightarrow *expr option* \Rightarrow *bool* ($- \mid - \vdash - \Rightarrow -$ [52,52,52,52] 51)

translations

$F \mid T \vdash e \Rightarrow e' == (F, T, e :: expr, e') \in run-expr$

inductive *run-expr intros*

REApp: $\llbracket F \mid T \vdash e1 \Rightarrow Some\ (EMem\ \tau\ f); (\tau, f, (\tau1, \tau2, e)) \in F;$

$\tau1 \in compty\ T; \tau2 \in compty\ T;$

$F \mid T \vdash e2 \Rightarrow Some\ e2'; F \mid T \vdash subst-expr\ e\ e2' \Rightarrow Some\ v \rrbracket$

$\implies F \mid T \vdash EApp\ e1\ e2 \Rightarrow Some\ v$
REAppE1: $\llbracket F \mid T \vdash e1 \Rightarrow a1; F \mid T \vdash e2 \Rightarrow a2; a1 = None \vee a2 = None \rrbracket$
 $\implies F \mid T \vdash EApp\ e1\ e2 \Rightarrow None$
REAppE2: $\llbracket F \mid T \vdash e1 \Rightarrow Some\ (EMem\ \tau\ f); \forall m. (\tau, f, m) \notin F \rrbracket$
 $\implies F \mid T \vdash EApp\ e1\ e2 \Rightarrow None$
REAppE3: $\llbracket F \mid T \vdash e1 \Rightarrow Some\ (EMem\ \tau\ f); (\tau, f, (\tau1, \tau2, e)) \in F;$
 $F \mid T \vdash e2 \Rightarrow Some\ e2'; F \mid T \vdash subst\text{-}expr\ e\ e2' \Rightarrow None \rrbracket$
 $\implies F \mid T \vdash EApp\ e1\ e2 \Rightarrow None$
REAppE4: $\llbracket F \mid T \vdash e1 \Rightarrow Some\ (EMem\ \tau\ f); (\tau, f, (\tau1, \tau2, e)) \in F;$
 $\tau1 \notin compty\ T \vee \tau2 \notin compty\ T \rrbracket$
 $\implies F \mid T \vdash EApp\ e1\ e2 \Rightarrow None$
REInt: $F \mid T \vdash EInt\ n \Rightarrow Some\ (EInt\ n)$
REObj: $\tau \in compty\ T \implies F \mid T \vdash EObj\ \tau \Rightarrow Some\ (EObj\ \tau)$
REMem: $F \mid T \vdash EMem\ \tau\ f \Rightarrow Some\ (EMem\ \tau\ f)$
REVarE: $F \mid T \vdash EVar \Rightarrow None$
REObjE: $\tau \notin compty\ T \implies F \mid T \vdash EObj\ \tau \Rightarrow None$

consts

expr-value :: *expr set*

inductive *expr-value* intros

EIntV[intro!]: *EInt* *n* \in *expr-value*

EObjV[intro!]: *EObj* $\tau \in$ *expr-value*

EMemV[intro!]: *EMem* $\tau\ f \in$ *expr-value*

lemma *run-expr-value-impl*: $F \mid T \vdash e \Rightarrow v \implies (\bigwedge e'. v = Some\ e' \implies e' \in expr\text{-}value)$

by (*induct rule*: *run-expr.induct*, *auto*)

lemma *run-expr-value*: $F \mid T \vdash e \Rightarrow Some\ v \implies v \in expr\text{-}value$

by (*simp add*: *run-expr-value-impl*)

consts

val :: *expr set*

inductive *val* intros

ValInt[intro!]: *EInt* *n* \in *val*

VObj[intro!]: $\tau \in residual\text{-}type \implies EObj\ \tau \in val$

VMem[intro!]: $\tau \in residual\text{-}type \implies EMem\ \tau\ f \in val$

lemma *subst-residual*:

$\llbracket e \in residual\text{-}expr; v \in residual\text{-}expr \rrbracket \implies subst\text{-}expr\ e\ v \in residual\text{-}expr$

by (*induct rule*: *residual-expr.induct*, *auto*)

lemma *val-residual*: $e \in val \implies e \in residual\text{-}expr$

by (*induct rule*: *val.induct*, *auto*)

inductive-cases *obj-resid*[*elim!*]:

EObj $\tau \in residual\text{-}expr$

inductive-cases *mem-resid*[*elim!*]:

EMem $\tau\ f \in residual\text{-}expr$

inductive-cases *mem-val*[*elim!*]:

EMem $\tau\ f \in val$

constdefs

resid-env :: *fun-set* \Rightarrow *bool*

resid-env *F* \equiv $(\forall \tau\ f\ \tau1\ \tau2\ e.$

$(\tau, f, (\tau1, \tau2, e)) \in F \implies e \in residual\text{-}expr)$

```

declare resid-env-def[simp]

lemma run-value-impl:  $F \mid T \vdash e \Rightarrow a$ 
   $\Rightarrow (\bigwedge v. \llbracket a = \text{Some } v; e \in \text{residual-expr}; \text{resid-env } F \rrbracket \Longrightarrow v \in \text{val})$ 
apply (induct rule: run-expr.induct)
prefer 2 apply simp
prefer 2 apply simp
prefer 2 apply simp
prefer 2 apply blast
prefer 2 apply blast
prefer 2 apply blast
prefer 2 apply simp
using val-residual subst-residual apply auto
done

lemma run-value:
   $\llbracket F \mid T \vdash e \Rightarrow \text{Some } v; e \in \text{residual-expr}; \text{resid-env } F \rrbracket$ 
   $\Longrightarrow v \in \text{val}$ 
by (simp add: run-value-impl)

constdefs
  fun-used-in :: expr  $\Rightarrow$  fun-used
  fun-used-in  $e \equiv \{ (\tau, f). (\text{EMem } \tau f, e) \in \text{subexpr} \}$ 
declare fun-used-in-def[simp]

constdefs
  fun-used-env :: fun-set  $\Rightarrow$  fun-used
  fun-used-env  $F \equiv \{ (\tau, f). (\exists \tau' f' \tau 1 \tau 2 e. (\tau', f', (\tau 1, \tau 2), e)) \in F$ 
     $\wedge (\tau, f) \in \text{fun-used-in } e) \}$ 
declare fun-used-env-def[simp]

constdefs
  ty-used-env :: fun-set  $\Rightarrow$  ty set
  ty-used-env  $F \equiv \{ \tau. (\exists \tau' f' \tau 1 \tau 2 e. (\tau', f', (\tau 1, \tau 2), e)) \in F$ 
     $\longrightarrow \tau \in \text{ty-used-in } e) \}$ 
declare ty-used-env-def[simp]

lemma subst-fun-used-sub:  $\bigwedge \tau f e 2. \text{fun-used-in } (\text{subst-expr } e 1 e 2)$ 
   $\subseteq \text{fun-used-in } e 1 \cup \text{fun-used-in } e 2$ 
by (induct e 1, auto, blast)

lemma track-used-fun:
   $F \mid T \vdash e \Rightarrow \text{Some } e'$ 
   $\Longrightarrow \text{fun-used-in } e' \subseteq \text{fun-used-in } e \cup \text{fun-used-env } F$ 
proof -
  { fix  $a$ 
    have  $F \mid T \vdash e \Rightarrow a$ 
       $\Longrightarrow (\bigwedge e'. a = \text{Some } e'$ 
         $\Longrightarrow \text{fun-used-in } e' \subseteq \text{fun-used-in } e \cup \text{fun-used-env } F)$ 
      apply (induct rule: run-expr.induct)
      defer
      apply blast
      apply blast
      apply blast
      apply blast
      apply blast
      apply blast
    }

```

```

apply blast
apply blast
apply blast
proof –
  fix  $T F \tau \tau 1 \tau 2 e e 1 e 2 e 2' f v e'$ 
  assume  $fe: (\tau, f, (\tau 1, \tau 2, e)) \in F$ 
  and  $ih2: \bigwedge e'. \text{Some } e 2' = \text{Some } e' \implies$ 
     $\text{fun-used-in } e' \subseteq \text{fun-used-in } e 2 \cup \text{fun-used-env } F$ 
  and  $ih3: \bigwedge e'. \text{Some } v = \text{Some } e' \implies$ 
     $\text{fun-used-in } e' \subseteq \text{fun-used-in } (\text{subst-expr } e e 2') \cup \text{fun-used-env } F$ 
  and  $ve: \text{Some } v = \text{Some } e'$ 
  from  $ih2$  have
     $ih2a: \text{fun-used-in } e 2' \subseteq \text{fun-used-in } e 2 \cup \text{fun-used-env } F$  by  $\text{simp}$ 
  from  $ve$   $ih3$  have
     $ih3a: \text{fun-used-in } e' \subseteq \text{fun-used-in } (\text{subst-expr } e e 2')$ 
     $\cup \text{fun-used-env } F$ 
    by  $\text{simp}$ 
  hence  $\text{fun-used-in } e' \subseteq \text{fun-used-in } e \cup \text{fun-used-in } e 2'$ 
     $\cup \text{fun-used-env } F$ 
  using  $\text{subst-fun-used-sub}$  apply  $\text{auto}$ 
  apply  $\text{blast}$ 
  done
  with  $ih2a$  have
     $\text{fun-used-in } e' \subseteq \text{fun-used-in } e \cup \text{fun-used-in } e 2$ 
     $\cup \text{fun-used-env } F$  by  $\text{blast}$ 
  with  $fe$ 
  show  $\text{fun-used-in } e' \subseteq \text{fun-used-in } (EApp e 1 e 2) \cup \text{fun-used-env } F$ 
    by  $\text{auto}$ 
  qed
} thus  $F \mid T \vdash e \implies \text{Some } e'$ 
   $\implies \text{fun-used-in } e' \subseteq \text{fun-used-in } e \cup \text{fun-used-env } F$ 
by  $\text{auto}$ 
qed

```

constdefs

```

 $\text{fun-used-defd} :: \text{fun-set} \implies \text{fun-used} \implies \text{bool}$ 
 $\text{fun-used-defd } F FU \equiv (\forall \tau f . (\tau, f) \in FU \longrightarrow (\exists tte. (\tau, f, tte) \in F))$ 
declare  $\text{fun-used-defd-def}[\text{simp}]$ 

```

consts $\text{pattern} :: \text{ty set}$

inductive pattern **intros**

```

 $PVar[\text{intro!}]: TVar i \in \text{pattern}$ 
 $PInt[\text{intro!}]: TInt \in \text{pattern}$ 
 $PTmId[\text{intro!}]: [\tau 1 \in \text{pattern}; \tau 2 \in \text{pattern}]$ 
   $\implies TmId t \tau 1 \tau 2 \in \text{pattern}$ 
 $PArrow[\text{intro!}]: [\tau 1 \in \text{pattern}; \tau 2 \in \text{pattern}]$ 
   $\implies TArrow \tau 1 \tau 2 \in \text{pattern}$ 

```

consts $\text{pat-equiv} :: \text{ty} \implies \text{ty} \implies \text{bool}$ (**infixl** $\doteq 61$)

constdefs

```

 $\text{no-dups} :: \text{tm-set} \implies \text{bool}$ 
 $\text{no-dups } T \equiv \forall \pi m \pi' m'. (\pi, m) \in T \wedge (\pi', m') \in T$ 
   $\wedge \pi \doteq \pi' \longrightarrow (\pi, m) = (\pi', m')$ 
declare  $\text{no-dups-def}[\text{simp}]$ 

```

constdefs

no-freevars :: *tm-set* \Rightarrow *bool*

no-freevars $T \equiv \forall \pi k n \tau. (\pi, k, n, \tau) \in T \longrightarrow FTV \tau \subseteq FTV \pi$

declare *no-freevars-def*[*simp*]

types *subst* = *nat* \Rightarrow *ty*

consts *subst-ty* :: *ty* \Rightarrow *subst* \Rightarrow *ty*

primrec

subst-ty (*TVar* *i*) *S* = *S i*

subst-ty (*TMem* τ *f*) *S* = *TMem* (*subst-ty* τ *S*) *f*

subst-ty (*TmId* *t* $\tau 1$ $\tau 2$) *S* = *TmId* *t* (*subst-ty* $\tau 1$ *S*) (*subst-ty* $\tau 2$ *S*)

subst-ty (*TArrow* $\tau 1$ $\tau 2$) *S* = *TArrow* (*subst-ty* $\tau 1$ *S*) (*subst-ty* $\tau 2$ *S*)

subst-ty *TInt* *S* = *TInt*

lemma *subst-nofree*:

\llbracket *subst-ty* π *S* = τ ; *FTV* τ = $\{\}$; $i \in FTV \pi$ \rrbracket

$\implies FTV (S i) = \{\}$

proof –

assume *a*: *subst-ty* π *S* = τ **and** *b*: *FTV* τ = $\{\}$ **and** *c*: $i \in FTV \pi$

{ **fix** π

have $\forall \tau S. \text{subst-ty } \pi S = \tau \wedge FTV \tau = \{\}$

$\longrightarrow (\forall i. i \in FTV \pi \longrightarrow FTV (S i) = \{\})$

by (*induct* π , *auto*)

} **with** *a b c* **show** *?thesis* **by** *blast*

qed

lemma *subst-nofree2-impl*:

$\forall \tau S. \text{subst-ty } \pi S = \tau \wedge (\forall i. i \in FTV \pi \longrightarrow FTV (S i) = \{\})$
 $\longrightarrow FTV \tau = \{\}$

(**is** *?P* π)

apply (*induct* π)

apply *simp*

apply *blast*

prefer 4 **apply** *simp*

prefer 3

apply *clarify*

apply (*erule-tac* $x = \text{subst-ty } \pi 1 S$ **in** *allE*)

apply (*erule-tac* $x = \text{subst-ty } \pi 2 S$ **in** *allE*)

apply (*erule-tac* $x = S$ **in** *allE*)

apply (*erule-tac* $x = S$ **in** *allE*)

apply *simp*

apply *clarify*

apply (*erule-tac* $x = \text{subst-ty } \pi S$ **in** *allE*)

apply (*erule-tac* $x = S$ **in** *allE*)

apply *simp*

apply *clarify*

apply (*erule-tac* $x = \text{subst-ty } \pi 1 S$ **in** *allE*)

apply (*erule-tac* $x = \text{subst-ty } \pi 2 S$ **in** *allE*)

apply (*erule-tac* $x = S$ **in** *allE*)

apply (*erule-tac* $x = S$ **in** *allE*)

apply *simp*

done

lemma *subst-nofree2*:

```

[[ subst-ty  $\pi S = \tau; \forall i. i \in FTV \pi \longrightarrow FTV (S i) = \{ \} ] ]
\implies FTV \tau = \{ \}$ 
```

using *subst-nofree2-impl* **apply** *blast* **done**

```

constdefs at-least-spec :: ty  $\Rightarrow$  ty  $\Rightarrow$  bool (infixl  $\leq$  61)
   $\tau 1 \leq \tau 2 \equiv (\exists S. \tau 1 = \text{subst-ty } \tau 2 S)$ 
declare at-least-spec-def[simp]

```

```

constdefs best-match :: tm-set  $\Rightarrow$  ty  $\Rightarrow$  ty  $\Rightarrow$  member  $\Rightarrow$  bool
  best-match T  $\tau$   $\pi$  m  $\equiv (\pi, m) \in T \wedge \tau \leq \pi$ 
   $\wedge (\forall \pi' m'. (\pi', m') \in T \wedge \tau \leq \pi' \longrightarrow \pi \leq \pi')$ 
declare best-match-def[simp]

```

```

defs pat-equiv-def:
   $\tau 1 \doteq (\tau 2 :: \text{ty}) \equiv \tau 1 \leq \tau 2 \wedge \tau 2 \leq \tau 1$ 
declare pat-equiv-def[simp]

```

```

defs lookup-def:
  lookup T  $\equiv \{ (\tau, k, a, \tau') . (\exists \pi \tau' S. \text{best-match } T \tau \pi (k, a, \tau') \wedge \text{subst-ty } \pi S = \tau \wedge \text{subst-ty } \tau' S = \tau') \}$ 
declare lookup-def[simp]

```

```

inductive-cases residual-type-inv[elim!]:
  TVar i  $\in$  residual-type
  TMem  $\tau$  f  $\in$  residual-type
  TArrow  $\tau 1$   $\tau 2$   $\in$  residual-type
  TmId t  $\tau 1$   $\tau 2$   $\in$  residual-type
  TInt  $\in$  residual-type

```

```

lemma equiv-tmid:
  assumes eq: TmId t  $\tau 1$   $\tau 2 \doteq \text{TmId } t' \tau 1' \tau 2'$ 
  shows  $t = t' \wedge \tau 1 \doteq \tau 1' \wedge \tau 2 \doteq \tau 2'$ 
proof –
  from eq have TmId t  $\tau 1$   $\tau 2 \leq \text{TmId } t' \tau 1' \tau 2'$ 
  apply (simp only: pat-equiv-def) done
  hence tt:  $t = t'$  by auto
  from eq have  $\tau 1 \doteq \tau 1' \wedge \tau 2 \doteq \tau 2'$  by auto
  with tt show ?thesis by simp
qed

```

```

lemma equiv-residual-eq:
   $\forall \tau'. \tau \doteq \tau' \wedge \tau \in \text{residual-type} \longrightarrow \tau = \tau'$ 
apply (induct  $\tau$ )
apply clarify
prefer 4 apply simp
apply clarify

```

```

apply clarify
apply (case-tac  $\tau'$ )
apply simp
apply simp
apply clarify
apply (frule equiv-tmid)
apply clarify
apply (erule-tac  $x = \text{ty1}$  in allE)

```

```

apply (erule-tac x=ty2 in allE)
apply (frule equiv-tmid)
apply simp
apply simp
apply simp

```

```

apply clarify
apply (case-tac τ')
apply simp
apply simp
apply simp
apply (erule-tac x=ty1 in allE)
apply (erule-tac x=ty2 in allE)
apply auto
done

```

```

lemma id-subst: τ = subst-ty τ (λ k. TVar k)
apply (induct τ) apply auto done

```

constdefs

```

wt-fun-env :: tm-set ⇒ fun-set ⇒ bool
wt-fun-env T F ≡ (∀ τ f τ1 τ2 e.
  (τ, f, (τ1,τ2,e)) ∈ F ∧ FTV τ = {}
  → (τ, Fun, f, TArrow τ1 τ2) ∈ T
  ∧ T | {} | Some τ1 ⊢ e : Some τ2
  ∧ τ1 ∈ compty T ∧ τ2 ∈ compty T)

```

```

declare wt-fun-env-def[simp]

```

```

lemma wf-ftv: [ T | V ⊢ τ wf; V = {} ] ⇒ FTV τ = {}
by (induct rule: wf-ty.induct, auto)

```

```

lemma eq-pat-eq: τ ≐ τ
apply (induct τ) using id-subst apply auto done

```

theorem

```

wt-expr-sound:
F | T ⊢ e ⇒ a
⇒ (∀ τ. T | {} | None ⊢ e : Some τ ∧
  fun-used-defd F (fun-used-in e ∪ fun-used-env F) ∧
  wt-fun-env T F ∧ no-dups T
  → (∃ v. a = Some v ∧ T | {} | None ⊢ v : Some τ))
(is ?A ⇒ ?IH F T e a)

```

proof (induct rule: run-expr.induct)

```

fix T F τ τ1 τ2 e e1 e2 e2' f v
assume IH1: ?IH F T e1 (Some (EMem τ f))
and F: (τ, f, (τ1, τ2, e)) ∈ F
and RE2: (F, T, e2, Some e2') ∈ run-expr
and IH2: ?IH F T e2 (Some e2')
and IH3: ?IH F T (subst-expr e e2') (Some v)
show ?IH F T (EApp e1 e2) (Some v)
proof clarify
fix τ'
assume TA: (T, {}, None, EApp e1 e2, Some τ') ∈ wt-expr
and FT: fun-used-defd F (fun-used-in (EApp e1 e2) ∪ fun-used-env F)
and WTF: wt-fun-env T F and nd: no-dups T
from TA obtain τ1' where TE1: T | {} | None ⊢ e1 : Some (TArrow τ1' τ')

```

and $TE2: T \mid \{\} \mid None \vdash e2 : Some \tau1'$ **by** (rule wt-app-inv, auto)
from FT **have**
 $FT1: fun-used-defd F (fun-used-in e1 \cup fun-used-env F)$ **by** auto

from $TE1 FT1 WTF nd IH1$
have $TV1: T \mid \{\} \mid None \vdash EMem \tau f : Some (TArrow \tau1' \tau')$ **by** blast
from FT **have**
 $FT2: fun-used-defd F (fun-used-in e2 \cup fun-used-env F)$ **by** auto
from $TE2 FT2 WTF nd IH2$ **have**
 $TV2: T \mid \{\} \mid None \vdash e2' : Some \tau1'$ **by** blast

from $TV1$ **have** $TT2: (\tau, Fun, f, TArrow \tau1' \tau') \in T$
apply (rule wt-mem-inv) **apply** auto **done**

from $TV1$ wt-mem-inv **have** $T \mid \{\} \vdash \tau wf$ **by** blast
with wf-ftv **have** $ftv: FTV \tau = \{\}$ **by** simp

— This is the important part. Need to know the body of the function is well typed.

from $F ftv WTF$ **have** $A: (\tau, Fun, f, TArrow \tau1 \tau2) \in T$
 $\wedge T \mid \{\} \mid Some \tau1 \vdash e : Some \tau2$
 $\wedge \tau1 \in compty T \wedge \tau2 \in compty T$
by (simp only: wt-fun-env-def, blast)

from A **have** $(\tau, Fun, f, TArrow \tau1 \tau2) \in T$ **by** simp
with $TT2 nd eq-pat-eq$
have $TArrow \tau1 \tau2 = TArrow \tau1' \tau'$
apply (simp only: no-dups-def) **apply** blast **done**
with A **have** $TE: T \mid \{\} \mid Some \tau1' \vdash e : Some \tau'$ **by** simp
from $TE TV2 subst-expr-type$ **have**
 $SE2: T \mid \{\} \mid None \vdash subst-expr e e2' : Some \tau'$ **by** simp
have $FT3:$
 $fun-used-defd F (fun-used-in (subst-expr e e2') \cup fun-used-env F)$
apply (simp only: fun-used-defd-def) **apply** clarify

proof –
fix $\tau' f'$
assume $(\tau', f') \in fun-used-in (subst-expr e e2') \cup fun-used-env F$
hence $a: (\tau', f') \in fun-used-in e \cup fun-used-in e2' \cup fun-used-env F$
using subst-fun-used-sub **apply** blast **done**
from $RE2$ **have** $e2p: fun-used-in e2' \subseteq fun-used-in e2 \cup fun-used-env F$
by (rule track-used-fun)
from $a e2p FT F$ **show** $\exists tte. (\tau', f', tte) \in F$ **by** (simp, blast)

qed
from $SE2 FT3 WTF nd IH3$
show $\exists va. Some v = Some va \wedge T \mid \{\} \mid None \vdash va : Some \tau'$ **by** blast

qed
next

fix $F T a1 a2 e1 e2$
assume $IH1: ?IH F T e1 a1$
and $IH2: ?IH F T e2 a2$
and $A12: a1 = None \vee a2 = None$
show $?IH F T (EApp e1 e2) None$
proof clarify
fix τ
assume $TA: (T, \{\}, None, EApp e1 e2, Some \tau) \in wt-expr$
and $FT: fun-used-defd F (fun-used-in (EApp e1 e2) \cup fun-used-env F)$

and WTF : $wt\text{-fun}\text{-env } T F$ **and** nd : $no\text{-dups } T$
from TA **obtain** $\tau 1$ **where** $TE1$: $T \mid \{\} \mid None \vdash e1 : Some (TArrow \tau 1 \tau)$
and $TE2$: $T \mid \{\} \mid None \vdash e2 : Some \tau 1$ **by** ($rule\ wt\text{-app}\text{-inv}, auto$)
note $A12$
moreover { **assume** $A1$: $a1 = None$
from FT **have**
 $FT1$: $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } e1 \cup fun\text{-used}\text{-env } F)$ **by** $auto$
from $TE1 FT1 WTF nd IH1$ **obtain** $v1$ **where** $A1V$: $a1 = Some v1$ **by** $blast$
from $A1V A1$
have $\exists v. None = Some v \wedge (T, \{\}, None, v, Some \tau) \in wt\text{-expr}$ **by** $simp$
} **moreover** { **assume** $A2$: $a2 = None$
from FT **have**
 $FT2$: $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } e2 \cup fun\text{-used}\text{-env } F)$ **by** $auto$
from $TE2 FT2 WTF nd IH2$ **obtain** $v2$ **where** $A2V$: $a2 = Some v2$ **by** $blast$
from $A2V A2$
have $\exists v. None = Some v \wedge (T, \{\}, None, v, Some \tau) \in wt\text{-expr}$ **by** $simp$
} **ultimately**
show $\exists v. None = Some v \wedge (T, \{\}, None, v, Some \tau) \in wt\text{-expr}$ **by** $blast$
qed
next
fix $T F \tau e1 e2 f$ — Case REAppE2, function lookup fails
assume $RE1$: $(F, T, e1, Some (EMem \tau f)) \in run\text{-expr}$
and F : $\forall m. (\tau, f, m) \notin F$
show $?IH F T (EApp e1 e2) None$
proof $clarify$
fix τ'
assume FT : $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } (EApp e1 e2) \cup fun\text{-used}\text{-env } F)$
from $RE1$ **have** $fun\text{-used}\text{-in } (EMem \tau f) \subseteq fun\text{-used}\text{-in } e1 \cup fun\text{-used}\text{-env } F$
by ($rule\ track\text{-used}\text{-fun}$)
hence $(\tau, f) \in (fun\text{-used}\text{-in } (EApp e1 e2) \cup fun\text{-used}\text{-env } F)$ **by** $auto$
with FT **have** $\exists tte. (\tau, f, tte) \in F$ **by** ($simp, blast$)
with F **have** $False$ **by** $auto$
thus $\exists v. None = Some v \wedge (T, \{\}, None, v, Some \tau') \in wt\text{-expr}$ **by** $simp$
qed
next
fix $T F \tau \tau 1 \tau 2 e e1 e2 e2' f$
assume $IH1$: $?IH F T e1 (Some (EMem \tau f))$
and F : $(\tau, f, (\tau 1, \tau 2, e)) \in F$
and $RE2$: $(F, T, e2, Some e2') \in run\text{-expr}$
and $IH2$: $?IH F T e2 (Some e2')$
and $IH3$: $?IH F T (subst\text{-expr } e e2') None$
show $?IH F T (EApp e1 e2) None$
proof $clarify$
fix τ'
assume TA : $(T, \{\}, None, EApp e1 e2, Some \tau') \in wt\text{-expr}$
and FT : $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } (EApp e1 e2) \cup fun\text{-used}\text{-env } F)$
and WTF : $wt\text{-fun}\text{-env } T F$ **and** nd : $no\text{-dups } T$
from TA **obtain** $\tau 1'$ **where** $TE1$: $T \mid \{\} \mid None \vdash e1 : Some (TArrow \tau 1' \tau')$
and $TE2$: $T \mid \{\} \mid None \vdash e2 : Some \tau 1'$ **by** ($rule\ wt\text{-app}\text{-inv}, auto$)
from FT **have**
 $FT1$: $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } e1 \cup fun\text{-used}\text{-env } F)$ **by** $auto$
from $TE1 FT1 WTF nd IH1$
have $TV1$: $T \mid \{\} \mid None \vdash EMem \tau f : Some (TArrow \tau 1' \tau')$ **by** $blast$

from FT **have**
 $FT2$: $fun\text{-used}\text{-defd } F (fun\text{-used}\text{-in } e2 \cup fun\text{-used}\text{-env } F)$ **by** $auto$

from *TE2 FT2 WTF nd IH2* have *TV2: T | {} | None ⊢ e2' : Some τ1' by blast*

from *TV1* have *TT2: (τ, Fun, f, TArrow τ1' τ') ∈ T*
 by (rule *wt-mem-inv, auto*)

from *TV1 wt-mem-inv* have *T | {} ⊢ τ wf by blast*

with *wf-ftv* have *ftv: FTV τ = {} by simp*

— This is the important part. Need to know the body of the function is well typed.

from *F ftv WTF* have *A: (τ, Fun, f, TArrow τ1 τ2) ∈ T*
 $\wedge T | \{ \} | \text{Some } \tau 1 \vdash e : \text{Some } \tau 2 \wedge \tau 1 \in \text{compty } T \wedge \tau 2 \in \text{compty } T$
 by (simp only: *wt-fun-env-def, blast*)

from *A* have (τ, Fun, f, TArrow τ1 τ2) ∈ T by simp

with *TT2 nd eq-pat-eq*

have *TArrow τ1 τ2 = TArrow τ1' τ'*

apply (simp only: *no-dups-def*) apply blast done

with *A* have *TE: T | {} | Some τ1' ⊢ e : Some τ' by simp*

from *TE TV2 subst-expr-type* have

SE2: T | {} | None ⊢ subst-expr e e2' : Some τ' by simp

have *FT3:*

fun-used-defd F (fun-used-in (subst-expr e e2') ∪ fun-used-env F)

apply (simp only: *fun-used-defd-def*) apply clarify

proof –

fix τ' f'

assume (τ', f') ∈ *fun-used-in (subst-expr e e2') ∪ fun-used-env F*

hence a: (τ', f') ∈ *fun-used-in e ∪ fun-used-in e2' ∪ fun-used-env F*

using *subst-fun-used-sub* apply blast done

from *RE2* have *e2p: fun-used-in e2' ⊆ fun-used-in e2 ∪ fun-used-env F*

by (rule *track-used-fun*)

from a *e2p FT F* show $\exists tte. (\tau', f', tte) \in F$ by (simp, blast)

qed

from *SE2 FT3 WTF nd IH3*

show $\exists v. \text{None} = \text{Some } v \wedge (T, \{ \}, \text{None}, v, \text{Some } \tau') \in \text{wt-expr}$ by blast

qed

next

fix *T F τ τ1 τ2 e e1 e2 f* — Case *REAppE4*, in/out types not complete

assume *IH1: ?IH F T e1 (Some (EMem τ f))*

and *F: (τ, f, (τ1, τ2, e)) ∈ (F::fun-set)*

and *T: τ1 ∉ compty T ∨ τ2 ∉ compty T*

show *?IH F T (EApp e1 e2) None*

proof clarify

fix τ'

assume *TA: (T, {}, None, EApp e1 e2, Some τ') ∈ wt-expr*

and *FT: fun-used-defd F (fun-used-in (EApp e1 e2) ∪ fun-used-env F)*

and *WTF: wt-fun-env T F* and *nd: no-dups T*

from *TA* obtain τ1' where *TE1: T | {} | None ⊢ e1 : Some (TArrow τ1' τ')*

and *TE2: T | {} | None ⊢ e2 : Some τ1' by (rule wt-app-inv, auto)*

from *FT* have

FT1: fun-used-defd F (fun-used-in e1 ∪ fun-used-env F) by auto

from *TE1 FT1 WTF nd IH1*

have *TV1: T | {} | None ⊢ EMem τ f : Some (TArrow τ1' τ')* by blast

from *TV1 wt-mem-inv* have *T | {} ⊢ τ wf by blast*

with *wf-ftv* have *FTV τ = {} by simp*

with *WTF F T* have *False* by (simp only: *wt-fun-env-def, blast*)

thus $\exists v. \text{None} = \text{Some } v \wedge (T, \{ \}, \text{None}, v, \text{Some } \tau') \in \text{wt-expr}$ by simp

```

qed
next
  fix T F n
  show ?IH F T (EInt n) (Some (EInt n)) by auto
next
  fix T F τ
  show ?IH F T (EObj τ) (Some (EObj τ)) by auto
next
  fix T F τ f
  show ?IH F T (EMem τ f) (Some (EMem τ f)) by auto
next
  fix T F
  from wt-var-inv show ?IH F T EVar None by blast
next
  fix F T τ
  assume nd: τ ∉ compty T
  show ?IH F T (EObj τ) None
  proof clarify
    fix τ'
    assume T | {} | None ⊢ EObj τ : Some τ'
    hence τ ∈ compty T by (rule wt-obj-inv, auto)
    with nd have False by simp
    thus ∃ v. None = Some v ∧ (T, {}, None, v, Some τ') ∈ wt-expr by simp
  qed
qed

```

```

datatype decl =
  Tm ty member
  | TmFun ty nat ty ty expr

```

```

consts subst-ty-expr :: expr ⇒ subst ⇒ expr

```

```

primrec

```

```

  subst-ty-expr EVar S = EVar
  subst-ty-expr (EInt n) S = EInt n
  subst-ty-expr (EObj τ) S = EObj (subst-ty τ S)
  subst-ty-expr (EMem τ f) S = EMem (subst-ty τ S) f
  subst-ty-expr (EApp e1 e2) S =
    EApp (subst-ty-expr e1 S) (subst-ty-expr e2 S)

```

```

constdefs

```

```

  tm-defined :: ty ⇒ tm-set ⇒ bool
  tm-defined τ T ≡ ∃ π' m'. π' ≐ τ ∧ (π', m') ∈ T

```

```

declare tm-defined-def[simp]

```

```

consts comp-prog :: (tm-set × fun-set × fun-used × decl list
  × (tm-set × fun-set) option) set

```

```

syntax (xsymbol)

```

```

  comp-prog :: tm-set ⇒ fun-set ⇒ fun-used ⇒ decl list ⇒ (tm-set × fun-set) option ⇒ bool (- | - | - ⊨ - ↓ -
[52,52,52,52,52] 51)

```

```

translations

```

```

  T | F | U ⊨ p ↓ a == (T,F,U,p,a) ∈ comp-prog

```

```

inductive comp-prog intros

```

```

  CNilP: T | F | {} ⊨ [] ↓ Some (T,F)
  CTmP: [¬ tm-defined (TmId t π1 π2) T;
    T | FTV (TmId t π1 π2) ⊢ τ ↓ Some τ'];

```

$(\text{insert } (TmId\ t\ \pi1\ \pi2, mk, mn, \tau')\ T) \mid F \mid U \models p1 \Downarrow ans$
 \Downarrow
 $\implies T \mid F \mid U \models (Tm\ (TmId\ t\ \pi1\ \pi2)\ (mk, mn, \tau))\#p1 \Downarrow ans$
CFunP: $\llbracket (\pi, Fun, f, TArrow\ \tau1\ \tau2) \in lookup\ T1;$
 $\forall \tau1a\ \tau2a\ ea. (\pi, f, (\tau1a, \tau2a, ea)) \notin F1;$
 $T1 \mid FTV\ \pi \vdash \tau1 \Downarrow Some\ \tau1';$
 $T1 \mid FTV\ \pi \vdash \tau2 \Downarrow Some\ \tau2';$
 $T1 \mid FTV\ \pi \vdash e \Downarrow e';$
 $\forall \tau. \tau \in (ty\text{-used-in}\ e' \cup \{\tau1', \tau2'\}) \wedge FTV\ \tau = \{\}$
 $\longrightarrow (\exists m. (\tau, m) \in lookup\ T1);$
 $T2 = \{ (\tau, m). \tau \in (ty\text{-used-in}\ e' \cup \{\tau1', \tau2'\}) \wedge FTV\ \tau = \{\}$
 $\wedge (\tau, m) \in lookup\ T1 \wedge \neg tm\text{-defined}\ \tau\ T1 \};$
 $insert\ (\pi, Fun, f, TArrow\ \tau1'\ \tau2')\ (T1 \cup T2)$
 $\mid FTV\ \pi \mid Some\ \tau1' \vdash e' : a;$
 $(FTV\ \pi \neq \{\} \wedge a = None) \vee a = Some\ \tau2';$
 $U2 = \{ (\tau, f). (EMem\ \tau\ f, e') \in subexpr \};$
 $insert\ (\pi, Fun, f, TArrow\ \tau1'\ \tau2')\ (T1 \cup T2)$
 $\mid insert\ (\pi, f, \tau1', \tau2', e')\ F1 \mid U1 \cup U2 \models p1 \Downarrow ans$
 \Downarrow
 $\implies T1 \mid F1 \mid U1 \models (TmFun\ \pi\ f\ \tau1\ \tau2\ e)\#p1 \Downarrow ans$
CIInstFunP:
 $\llbracket \forall tte. (\tau, f, tte) \notin F1;$
 $(\tau, f) \notin U1;$
 $best\text{-match}\ T1\ \tau\ \pi\ (Fun, f, TArrow\ \tau1\ \tau2);$
 $(\pi, n, (\tau1, \tau2, e)) \in F1;$
 $subst\text{-ty}\ \pi\ S = \tau;$
 $T1 \mid \{\} \vdash subst\text{-ty}\ \tau1\ S \Downarrow Some\ \tau1';$
 $T1 \mid \{\} \vdash subst\text{-ty}\ \tau2\ S \Downarrow Some\ \tau2';$
 $T1 \mid \{\} \vdash (subst\text{-ty}\text{-expr}\ e\ S) \Downarrow e';$
 $\forall \tau. \tau \in (ty\text{-used-in}\ e' \cup \{\tau1', \tau2'\}) \wedge FTV\ \tau = \{\}$
 $\longrightarrow (\exists m. (\tau, m) \in lookup\ T1);$
 $T2 = \{ (\tau, m). \tau \in (ty\text{-used-in}\ e' \cup \{\tau1', \tau2'\}) \wedge FTV\ \tau = \{\}$
 $\wedge (\tau, m) \in lookup\ T1 \wedge \neg tm\text{-defined}\ \tau\ T1 \};$
 $insert\ (\tau, Fun, f, TArrow\ \tau1'\ \tau2')\ (T1 \cup T2)$
 $\mid \{\} \mid Some\ \tau1' \vdash e' : Some\ \tau2';$
 $U2 = \{ (\tau, f). (EMem\ \tau\ f, e') \in subexpr \};$
 $insert\ (\tau, Fun, f, TArrow\ \tau1'\ \tau2')\ (T1 \cup T2)$
 $\mid insert\ (\tau, f, \tau1', \tau2', e')\ F1 \mid U1 \cup U2 \models \square \Downarrow ans$
 \Downarrow
 $\implies T1 \mid F1 \mid insert\ (\tau, f)\ U1 \models \square \Downarrow ans$
CFunDefdP:
 $\llbracket (\tau, f, tte) \in F; (\tau, f) \notin U; T \mid F \mid U \models \square \Downarrow ans \rrbracket$
 $\implies T \mid F \mid insert\ (\tau, f)\ U \models \square \Downarrow ans$

constdefs

$fun\text{-used-invar} :: fun\text{-set} \Rightarrow fun\text{-used} \Rightarrow bool$
 $fun\text{-used-invar}\ F\ U \equiv$
 $(\forall \tau\ f. (\tau, f) \in fun\text{-used-env}\ F$
 $\longrightarrow (\exists tte. (\tau, f, tte) \in F) \vee (\tau, f) \in U)$

declare $fun\text{-used-invar-def}$ [simp]

lemma lookup-freevars:

assumes $l: (\tau, k, n, \tau') \in lookup\ T$
and $ft: FTV\ \tau = \{\}$

and nT : *no-freevars* T
shows $FTV \tau' = \{\}$
proof –
from l **obtain** $\pi \tau'' S$ **where**
 bm : *best-match* $T \tau \pi (k, n, \tau'')$
and ps : *subst-ty* $\pi S = \tau$
and tps : *subst-ty* $\tau'' S = \tau'$ **by** *auto*
from bm **have** $p1T$: $(\pi, (k, n, \tau'')) \in T$ **by** *simp*
with nT **have** $FTV \tau'' \subseteq FTV \pi$ **by** *simp*
with $ft ps$ *subst-nofree*
have $\forall i. i \in FTV \tau'' \longrightarrow FTV (S i) = \{\}$
apply *auto* **done**
with tps *subst-nofree2* **show** $FTV \tau' = \{\}$
apply *simp* **done**
qed

lemma *wf-ty-ext-tm*:
 $\llbracket T \mid V \vdash \tau \text{ wf}; T \subseteq T' \rrbracket \Longrightarrow T' \mid V \vdash \tau \text{ wf}$
apply (*induct rule: wf-ty.induct*)
apply *auto*
apply (*simp add: tms-def*)
apply *blast*
done

lemma *wt-expr-ext-tm*:
 $\llbracket T \mid V \mid \text{Some } \tau 1 \vdash e : \text{Some } \tau 2; T \subseteq T' \rrbracket$
 $\Longrightarrow T' \mid V \mid \text{Some } \tau 1 \vdash e : \text{Some } \tau 2$
apply (*induct rule: wt-expr.induct*)
apply *force*
apply *force*
apply *force*
apply *force*
prefer 2 **apply** *force*
prefer 3 **apply** *force*
proof –
fix $T V \tau \tau 1 \tau 2 t$
assume wft : *wf-ty* $T V (TmId t \tau 1 \tau 2)$
and $ft1$: $FTV \tau 1 = \{\}$ **and** $ft2$: $FTV \tau 2 = \{\}$
and c : $TmId t \tau 1 \tau 2 \in \text{compty } T$
and ttp : $T \subseteq T'$
from $wft ttp$ **have** $wft2$: $T' \mid V \vdash (TmId t \tau 1 \tau 2) \text{ wf}$
by (*rule wf-ty-ext-tm*)
from $c ttp$ **have** $c2$: $TmId t \tau 1 \tau 2 \in \text{compty } T'$
apply *auto* **done**
from $wft2 ft1 ft2 c2$
show $(T', V, \tau, EObj (TmId t \tau 1 \tau 2), \text{Some } (TmId t \tau 1 \tau 2)) \in \text{wt-expr}$
apply (*rule TEObj*) **done**
next
fix $T V \tau \tau 1 \tau 2 \tau 3 f$
assume wft : *wf-ty* $T V \tau$ **and** tT : $(\tau, Fun, f, TArrow \tau 1 \tau 2) \in T$
and ttp : $T \subseteq T'$
from $wft ttp$ **have** $wft2$: $T' \mid V \vdash \tau \text{ wf}$ **by** (*rule wf-ty-ext-tm*)
from $tT ttp$ **have** $tT2$: $(\tau, Fun, f, TArrow \tau 1 \tau 2) \in T'$
apply *auto* **done**
from $wft2 tT2$
show $(T', V, \tau 3, EMem \tau f, \text{Some } (TArrow \tau 1 \tau 2)) \in \text{wt-expr}$

apply (rule TEMem) done
qed

lemma ty-eval-ftv:

$T \mid V \vdash \tau \Downarrow a$
 $\implies (\bigwedge \tau'. \llbracket a = \text{Some } \tau' \rrbracket \implies FTV \tau' \subseteq V)$
 apply (induct rule: ty-eval.induct)
 apply force+
 done

theorem compile-preserves:

$T \mid F \mid U \models p \Downarrow a$
 $\implies (\forall T' F'. a = \text{Some } (T', F') \wedge \text{fun-used-invar } F U \wedge \text{wt-fun-env } T F$
 $\quad \wedge \text{no-dups } T \wedge \text{no-freevars } T$
 $\quad \longrightarrow \text{fun-used-invar } F' \{ \} \wedge \text{wt-fun-env } T' F'$
 $\quad \wedge \text{no-dups } T' \wedge \text{no-freevars } T')$
 (is ?A \implies ?IH $T F U a$)

proof (induct rule: comp-prog.induct)

fix F T
 show ?IH T F { } (Some (T, F)) by blast

next

fix F T U $\pi 1 \pi 2 \tau \tau'$ ans mk mn p1 t
 assume tnd: $\neg \text{tm-defined } (TmId t \pi 1 \pi 2) T$
 and mtv: $T \mid FTV (TmId t \pi 1 \pi 2) \vdash \tau \Downarrow (\text{Some } \tau')$
 and cp1: $(\text{insert } (TmId t \pi 1 \pi 2, (mk, mn, \tau')) T) \mid F \mid U \models p1 \Downarrow \text{ans}$
 and ih: ?IH $(\text{insert } (TmId t \pi 1 \pi 2, mk, mn, \tau') T) F U \text{ans}$
 show ?IH T F U ans
 apply (rule allI) apply (rule allI) apply (rule impI) apply (erule conjE)+
 proof -
 fix T' F'
 assume a: $\text{ans} = \text{Some } (T', F')$ and fu: $\text{fun-used-invar } F U$
 and wtf: $\text{wt-fun-env } T F$ and nd: $\text{no-dups } T$ and nf: $\text{no-freevars } T$
 let ?TM = $(TmId t \pi 1 \pi 2, (mk, mn, \tau'))$
 let ?T = $\text{insert } ?TM T$

have nd: $\text{no-dups } ?T$

apply (simp only: no-dups-def)
 apply clarify

proof -

fix $\pi a aa b \pi' ab ac ba$
 assume pT: $(\pi, a, aa, b) \in \text{insert } (TmId t \pi 1 \pi 2, mk, mn, \tau') T$
 and ppT: $(\pi', ab, ac, ba) \in \text{insert } (TmId t \pi 1 \pi 2, mk, mn, \tau') T$
 and ppp: $\pi \doteq \pi'$

have $(\pi, a, aa, b) = ?TM \vee (\pi, a, aa, b) \neq ?TM$ by simp

moreover { assume ptm: $(\pi, a, aa, b) = ?TM$

have $(\pi', ab, ac, ba) = ?TM \vee (\pi', ab, ac, ba) \neq ?TM$ by simp

moreover { assume pptm: $(\pi', ab, ac, ba) = ?TM$

from ptm pptm have $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$

by simp

} moreover { assume pptm: $(\pi', ab, ac, ba) \neq ?TM$

with ppT have ppt: $(\pi', ab, ac, ba) \in T$ by auto

from ptm ppp have $\pi' \doteq TmId t \pi 1 \pi 2$ by simp

with tnd have $(\pi', ab, ac, ba) \notin T$

apply auto done

with ppt have False by simp

hence $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$ **by simp**
 } ultimately have $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$ **by blast**
 } moreover { assume $ptm: (\pi, a, aa, b) \neq ?TM$
 have $(\pi', ab, ac, ba) = ?TM \vee (\pi', ab, ac, ba) \neq ?TM$ **by simp**
 moreover { assume $pptm: (\pi', ab, ac, ba) = ?TM$
 from $ptm pT$ have $pt: (\pi, a, aa, b) \in T$ **by auto**
 from $pptm ppp$ have $\pi \doteq TmId t \pi1 \pi2$ **by simp**
 with tnd have $(\pi, a, aa, b) \notin T$ **by auto**
 with pt have $False$ **by simp**
 hence $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$ **by simp**
 } moreover { assume $pptm: (\pi', ab, ac, ba) \neq ?TM$
 from $ptm pT$ have $pt: (\pi, a, aa, b) \in T$ **by auto**
 from $pptm ppT$ have $ppt: (\pi', ab, ac, ba) \in T$ **by auto**
 from $pt ppt ppp nd$
 have $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$
 by (*simp only: no-dups-def, blast*)
 } ultimately have $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$ **by blast**
 } ultimately show $\pi = \pi' \wedge (a, aa, b) = (ab, ac, ba)$ **by blast**
qed

have $nf: no-freevars ?T$

proof –

from $mtv nf ty-eval-ftv$

have $FTV \tau' \subseteq FTV (TmId t \pi1 \pi2)$ **by simp**

with nf show *?thesis* **by simp**

qed

have $wt-fun-env ?T F$

apply (*simp only: wt-fun-env-def*)

apply (*rule allI*)**+** **apply** (*rule impI*) **apply** (*erule conjE*)

proof –

fix $\tau'' f \tau1 \tau2 e$

assume $tff: (\tau'', f, (\tau1, \tau2, e)) \in F$

and $ftv: FTV \tau'' = \{\}$

from $tff ftv wtf$ **have** $(\tau'', Fun, f, TArrow \tau1 \tau2) \in T$

by (*simp only: wt-fun-env-def, blast*)

hence $tft: (\tau'', Fun, f, TArrow \tau1 \tau2) \in ?T$ **by simp**

from $tff ftv wtf$ **have** $T \mid \{\} \mid Some \tau1 \vdash e : Some \tau2$

by (*simp only: wt-fun-env-def, blast*)

hence $e: ?T \mid \{\} \mid Some \tau1 \vdash e : Some \tau2$

apply (*rule wt-expr-ext-tm*)

apply *auto done*

from $tff ftv wtf$ **have** $nT: \tau1 \in compty T$

by (*simp only: wt-fun-env-def, blast*)

hence $t1c: \tau1 \in compty ?T$

apply (*simp add: compty-def*)

apply *blast done*

from $tff ftv wtf$ **have** $nT: \tau2 \in compty T$

by (*simp only: wt-fun-env-def, blast*)

hence $t2c: \tau2 \in compty ?T$

apply (*simp add: compty-def*)

apply *blast done*

```

from tft e nd nf t1c t2c
show  $(\tau'', Fun, f, TArrow \tau1 \tau2) \in ?T$ 
       $\wedge ?T \mid \{\} \mid Some \tau1 \vdash e : Some \tau2$ 
       $\wedge \tau1 \in compty ?T \wedge \tau2 \in compty ?T$ 
apply clarify done
qed
with a nd nf fu ih show  $fun-used-invar F' \{\} \wedge wt-fun-env T' F'$ 
       $\wedge no-dups T' \wedge no-freevars T'$  by blast
qed
next
fix F1 T T1 T2 T3 U1 U2  $\pi$   $\tau1$   $\tau1'$   $\tau2$   $\tau2'$  a ans e e' f p1
assume  $(\pi, Fun, f, TArrow \tau1 \tau2) \in lookup T1$ 
and pf1:  $\forall \tau1a \tau2a ea. (\pi, f, \tau1a, \tau2a, ea) \notin F1$ 
and tv1: ty-eval T1 (FTV  $\pi$ )  $\tau1$  (Some  $\tau1'$ )
and tv2: ty-eval T1 (FTV  $\pi$ )  $\tau2$  (Some  $\tau2'$ )
and  $(T1, FTV \pi, e, e') \in comp-expr$ 
and inst:  $\forall \tau. \tau \in ty-used-in e' \cup \{\tau1', \tau2'\} \wedge FTV \tau = \{\} \longrightarrow$ 
       $(\exists m. (\tau, m) \in lookup T1)$ 
and t2:  $T2 = \{(\tau, m). \tau \in (ty-used-in e' \cup \{\tau1', \tau2'\}) \wedge$ 
       $FTV \tau = \{\} \wedge (\tau, m) \in lookup T1 \wedge \neg tm-defined \tau T1\}$ 
and epwt: (insert ( $\pi, Fun, f, TArrow \tau1' \tau2'$ ) (T1  $\cup$  T2),
       $FTV \pi, Some \tau1', e', a) \in wt-expr$ 
and fta:  $FTV \pi \neq \{\} \wedge a = None \vee a = Some \tau2'$ 
and u2:  $U2 = \{(\tau, f). (EMem \tau f, e') \in subexpr\}$ 
and comp-prog (insert ( $\pi, Fun, f, TArrow \tau1' \tau2'$ ) (T1  $\cup$  T2))
       $(insert (\pi, f, \tau1', \tau2', e') F1) (U1 \cup U2) p1 ans$ 
and ih: ?IH (insert ( $\pi, Fun, f, TArrow \tau1' \tau2'$ ) (T1  $\cup$  T2))
       $(insert (\pi, f, \tau1', \tau2', e') F1) (U1 \cup U2) ans$ 
show ?IH T1 F1 U1 ans
apply (rule allI) apply (rule allI) apply (rule impI) apply (erule conjE)+
proof -
fix T' F'
assume a: ans = Some (T',F') and f1u1: fun-used-invar F1 U1
and wtf: wt-fun-env T1 F1 and nd: no-dups T1 and nf: no-freevars T1
let ?T = (insert ( $\pi, Fun, f, TArrow \tau1' \tau2'$ ) (T1  $\cup$  T2))
let ?F = (insert ( $\pi, f, \tau1', \tau2', e')$  F1)
let ?U = (U1  $\cup$  U2)

have fu: fun-used-invar ?F ?U
apply (simp only: fun-used-invar-def) apply (rule allI)+
apply (rule impI)
proof -
fix  $\tau' f'$ 
assume tf:  $(\tau', f') \in fun-used-env ?F$ 
let ?E =  $(\exists tte. (\tau', f', tte) \in ?F) \vee (\tau', f') \in U1 \cup U2$ 
from tf have  $(\tau', f') \in fun-used-in e' \cup fun-used-env F1$ 
by (simp, blast)
moreover { assume  $(\tau', f') \in fun-used-in e'$ 
with u2 have  $(\tau', f') \in U1 \cup U2$  by simp
hence ?E by simp
} moreover { assume  $(\tau', f') \in fun-used-env F1$ 
with f1u1 have
      tte:  $(\exists tte. (\tau', f', tte) \in F1) \vee (\tau', f') \in U1$  by simp
from tte obtain tte where
      tf1:  $(\tau', f', tte) \in F1 \vee (\tau', f') \in U1$  by auto
moreover { assume  $(\tau', f', tte) \in F1$ 

```

```

    hence  $(\tau', f', tte) \in \text{insert } (\pi, f, \tau 1', \tau 2', e')$  F1 by simp
    hence ?E by blast
  } moreover { assume  $(\tau', f') \in U1$  hence ?E by simp
  } ultimately have ?E by auto
} ultimately show ?E by blast
qed

```

```

have wtenv: wt-fun-env ?T ?F
apply (simp only: wt-fun-env-def)
apply (rule allI) + apply (rule impI) apply (erule conjE)
proof –
fix  $\tau$   $g$   $t1$   $t2$   $e$ 
assume tf:  $(\tau, g, t1, t2, e) \in ?F$  and ftv: FTV  $\tau = \{\}$ 

```

```

have tft:  $(\tau, Fun, g, TArrow\ t1\ t2) \in ?T$ 
proof –
from tf
have  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 
   $\vee (\tau, g, t1, t2, e) \in F1$  apply auto done
moreover { assume  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 
hence  $(\tau, Fun, g, TArrow\ t1\ t2) \in ?T$  by simp
} moreover { assume tf1:  $(\tau, g, t1, t2, e) \in F1$ 
from tf1 ftv wtf have  $(\tau, Fun, g, TArrow\ t1\ t2) \in T1$ 
by (simp only: wt-fun-env-def, blast)
hence  $(\tau, Fun, g, TArrow\ t1\ t2) \in ?T$  by auto
} ultimately show  $(\tau, Fun, g, TArrow\ t1\ t2) \in ?T$  by blast
qed

```

```

have e: ?T |  $\{\}$  | Some  $t1 \vdash e$  : Some  $t2$ 
proof –
from tf have  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 
   $\vee (\tau, g, t1, t2, e) \in F1$  apply auto done
moreover { assume tp:  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 
from ftv tp eput fta have ?T |  $\{\}$  | Some  $t1 \vdash e$  : Some  $t2$  by simp
} moreover { assume tf1:  $(\tau, g, t1, t2, e) \in F1$ 
from tf1 ftv wtf have T1 |  $\{\}$  | Some  $t1 \vdash e$  : Some  $t2$ 
by (simp only: wt-fun-env-def, blast)
hence ?T |  $\{\}$  | Some  $t1 \vdash e$  : Some  $t2$ 
apply (rule wt-expr-ext-tm)
apply auto done
} ultimately show ?T |  $\{\}$  | Some  $t1 \vdash e$  : Some  $t2$  by blast
qed

```

```

have tc:  $t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$ 
proof –
from tf have  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 
   $\vee (\tau, g, t1, t2, e) \in F1$  apply auto done
moreover { assume tp:  $(\tau, g, t1, t2, e) = (\pi, f, \tau 1', \tau 2', e')$ 

from tp tv1 ftv eval-type-residual
have t1r:  $t1 \in \text{residual-type}$  by blast
hence t1c: FTV  $t1 = \{\}$  by (rule residual-closed)
from tp tv2 ftv eval-type-residual
have t2r:  $t2 \in \text{residual-type}$  by blast
hence t2c: FTV  $t2 = \{\}$  by (rule residual-closed)

```

```

from  $tp$   $t1c$  have  $ft1p$ :  $FTV \tau 1' = \{\}$  by simp
have  $\tau 1' \in (ty-used-in\ e' \cup \{\tau 1', \tau 2'\})$  apply simp done
with  $tp$  have  $t1u$ :  $t1 \in (ty-used-in\ e' \cup \{\tau 1', \tau 2'\})$  apply simp done
from  $ft1p$   $t1u$  inst have  $exm$ :  $\exists m. (\tau 1', m) \in lookup\ T1$  by auto
from  $exm$  obtain  $m1$  where  $(\tau 1', m1) \in lookup\ T1$  apply blast done
with  $tp$  have  $t1l$ :  $(t1, m1) \in lookup\ T1$  by simp
have  $\neg tm-defined\ t1\ T1 \vee (tm-defined\ t1\ T1)$  by auto
moreover { assume  $ndt1$ :  $\neg tm-defined\ t1\ T1$ 
  from  $t1u$   $t1c$   $t1l$   $ndt1$   $t2$  have  $t1T2$ :  $(t1, m1) \in T2$ 
  apply simp done
  with  $t1r$  have  $t1 \in compty\ ?T$ 
  apply (simp only: compty-def) apply blast done
} moreover { assume  $tmd$ :  $tm-defined\ t1\ T1$ 
  from  $tmd$  obtain  $p1$   $m1'$  where  $pt1$ :  $t1 \doteq p1$  and  $pm1$ :  $(p1, m1') \in T1$ 
  apply auto apply blast done
  from  $t1r$   $pt1$  equiv-residual-eq have  $pt1eq$ :  $t1 = p1$ 
  apply blast done
  with  $pm1$  have  $(t1, m1') \in T1$  by simp
  with  $t1r$  have  $t1 \in compty\ ?T$ 
  apply (simp only: compty-def) apply blast done
} ultimately have  $ct1$ :  $t1 \in compty\ ?T$  by blast

from  $tp$   $t2c$  have  $ft2p$ :  $FTV \tau 2' = \{\}$  by simp
have  $\tau 2' \in (ty-used-in\ e' \cup \{\tau 1', \tau 2'\})$  apply simp done
with  $tp$  have  $t2u$ :  $t2 \in (ty-used-in\ e' \cup \{\tau 1', \tau 2'\})$  apply simp done
from  $ft2p$   $t2u$  inst have  $exm2$ :  $\exists m. (\tau 2', m) \in lookup\ T1$  by auto
from  $exm2$  obtain  $m2$  where  $(\tau 2', m2) \in lookup\ T1$  apply blast done
with  $tp$  have  $t2l$ :  $(t2, m2) \in lookup\ T1$  by simp
have  $\neg tm-defined\ t2\ T1 \vee (tm-defined\ t2\ T1)$  by auto
moreover { assume  $ndt2$ :  $\neg tm-defined\ t2\ T1$ 
  from  $t2u$   $t2c$   $t2l$   $ndt2$   $t2$  have  $t2T2$ :  $(t2, m2) \in T2$  by simp
  with  $t2r$  have  $t2 \in compty\ ?T$ 
  apply (simp only: compty-def) apply blast done
} moreover { assume  $tmd$ :  $tm-defined\ t2\ T1$ 
  from  $tmd$  obtain  $p1$   $m1'$  where  $pt1$ :  $t2 \doteq p1$  and  $pm1$ :  $(p1, m1') \in T1$ 
  apply auto apply blast done
  from  $t2r$   $pt1$  equiv-residual-eq have  $pt1eq$ :  $t2 = p1$ 
  apply blast done
  with  $pm1$  have  $(t2, m1') \in T1$  by simp
  with  $t2r$  have  $t2 \in compty\ ?T$ 
  apply (simp only: compty-def) apply blast done
} ultimately have  $ct2$ :  $t2 \in compty\ ?T$  by blast
from  $ct1$   $ct2$  have  $t1 \in compty\ ?T \wedge t2 \in compty\ ?T$  by blast
} moreover { assume  $tf1$ :  $(\tau, g, t1, t2, e) \in F1$ 
from  $tf1$   $ftv$   $wtf$  have  $tc1$ :  $t1 \in compty\ T1 \wedge t2 \in compty\ T1$ 
  by (simp only: wt-fun-env-def, blast)
  hence  $t1 \in compty\ ?T \wedge t2 \in compty\ ?T$  by auto
} ultimately show  $t1 \in compty\ ?T \wedge t2 \in compty\ ?T$  by blast
qed

```

```

from  $tft$   $e$   $tc$ 
show  $(\tau, Fun, g, TArrow\ t1\ t2) \in ?T$ 
   $\wedge (?T, \{\}, Some\ t1, e, Some\ t2) \in wt-expr$ 
   $\wedge t1 \in compty\ ?T \wedge t2 \in compty\ ?T$ 
apply blast done
qed

```

```

have ndT: no-dups ?T sorry

have nfT: no-freevars ?T
proof -
  have nf2: no-freevars T2
  apply (simp only: no-freevars-def) apply clarify
  proof -
    fix  $\pi$   $k$   $n$   $\tau$   $x$ 
    assume pt2:  $(\pi, k, n, \tau) \in T2$  and xft:  $x \in FTV \tau$ 
    from pt2 t2 have l:  $(\pi, k, n, \tau) \in \text{lookup } T1$  by simp
    from pt2 t2 have pf:  $FTV \pi = \{\}$  by simp
    from l pf nf have FTV  $\tau = \{\}$ 
    apply (rule lookup-freevars) done
    with xft have False by simp
    thus  $x \in FTV \pi$  by simp
  qed
  from nf nf2 have nf12: no-freevars  $(T1 \cup T2)$  by auto

  from tv1 nf ty-eval-ftv
  have ft1:  $FTV \tau1' \subseteq FTV \pi$  by simp
  from tv2 nf ty-eval-ftv
  have ft2:  $FTV \tau2' \subseteq FTV \pi$  by simp
  from ft1 ft2 have FTV  $(TArrow \tau1' \tau2') \subseteq FTV \pi$  by simp

  with nf12 show ?thesis by simp
  qed

  from a fu wtenv ndT nfT ih show fun-used-invar  $F' \{\}$   $\wedge$  wt-fun-env  $T' F'$ 
   $\wedge$  no-dups  $T' \wedge$  no-freevars  $T'$ 
  apply blast done
  qed
next
fix F1 S T1 T2 U1 U2  $\pi$   $\tau$   $\tau'$   $\tau1$   $\tau1'$   $\tau2$   $\tau2'$  ans e e' f n
assume tf1:  $\forall tte. (\tau, f, tte) \notin F1$ 
and tfu1:  $(\tau, f) \notin U1$ 
and bm: best-match T1  $\tau \pi$   $(Fun, f, \tau')$ 
and pf1:  $(\pi, n, \tau1, \tau2, e) \in F1$ 
and sps: subst-ty  $\pi S = \tau$ 
and tv1: ty-eval T1  $\{\}$   $(\text{subst-ty } \tau1 S)$   $(Some \tau1')$ 
and tv2: ty-eval T1  $\{\}$   $(\text{subst-ty } \tau2 S)$   $(Some \tau2')$ 
and  $(T1, \{\}, \text{subst-ty-expr } e S, e') \in \text{comp-expr}$ 
and inst:  $\forall \tau. \tau \in \text{ty-used-in } e' \cup \{\tau1', \tau2'\} \wedge FTV \tau = \{\} \longrightarrow$ 
 $(\exists m. (\tau, m) \in \text{lookup } T1)$ 
and t2: T2 =
 $\{(\tau, m).$ 
 $\tau \in \text{ty-used-in } e' \cup \{\tau1', \tau2'\} \wedge$ 
 $FTV \tau = \{\} \wedge (\tau, m) \in \text{lookup } T1 \wedge \neg \text{tm-defined } \tau T1\}$ 
and epwt:  $(\text{insert } (\tau, Fun, f, TArrow \tau1' \tau2') (T1 \cup T2), \{\}, Some \tau1', e', Some \tau2')$ 
 $\in \text{wt-expr}$ 
and u2:  $U2 = \{(\tau, f). (EMem \tau f, e') \in \text{subexpr}\}$ 
and comp-prog  $(\text{insert } (\tau, Fun, f, TArrow \tau1' \tau2') (T1 \cup T2))$ 
 $(\text{insert } (\tau, f, \tau1', \tau2', e') F1) (U1 \cup U2) \square$  ans
and ih: ?IH  $(\text{insert } (\tau, Fun, f, TArrow \tau1' \tau2') (T1 \cup T2))$ 
 $(\text{insert } (\tau, f, \tau1', \tau2', e') F1) (U1 \cup U2)$  ans
show ?IH T1 F1  $(\text{insert } (\tau, f) U1)$  ans

```

```

apply (rule allI)+ apply (rule impI) apply (erule conjE)+
proof -
  fix T' F'
  assume a: ans = Some (T',F') and f1u1: fun-used-invar F1 (insert (τ,f) U1)
    and wtf: wt-fun-env T1 F1 and nd: no-dups T1 and nf: no-freevars T1
  let ?F = (insert (τ, f, τ1', τ2', e') F1)
  let ?T = (insert (τ, Fun, f, TArrow τ1' τ2') (T1 ∪ T2))

  have fu: fun-used-invar ?F (U1 ∪ U2)
    apply (simp only: fun-used-invar-def) apply (rule allI)+ apply (rule impI)
  proof -
    fix τ' f'
    assume tf: (τ', f') ∈ fun-used-env ?F
    let ?E = (∃ tte. (τ', f', tte) ∈ ?F) ∨ (τ', f') ∈ U1 ∪ U2
    from tf have (τ', f') ∈ fun-used-in e' ∪ fun-used-env F1
      by (simp, blast)
    moreover { assume (τ', f') ∈ fun-used-in e'
      with u2 have (τ', f') ∈ U1 ∪ U2 by simp
      hence ?E by simp
    } moreover { assume (τ', f') ∈ fun-used-env F1
      with f1u1 have
        tte: (∃ tte. (τ',f',tte) ∈ F1) ∨ (τ', f') ∈ (insert (τ,f) U1) by simp
      from tte obtain tte where
        tf1: (τ',f',tte) ∈ F1 ∨ (τ', f') ∈ (insert (τ,f) U1) by auto
      moreover { assume (τ',f',tte) ∈ F1
        hence (τ',f',tte) ∈ insert (τ, f, τ1', τ2', e') F1 by simp
        hence ?E apply blast done
      } moreover { assume (τ', f') ∈ (insert (τ,f) U1)
        hence (τ', f') = (τ,f) ∨ (τ',f') ∈ U1 by auto
        moreover { assume (τ', f') = (τ,f)
          with tf1 have ?E apply blast done
        } moreover { assume (τ',f') ∈ U1 hence ?E by auto }
      ultimately have ?E by blast
    } ultimately have ?E by blast
  } ultimately show ?E by blast
qed

```

```

have wtenv: wt-fun-env ?T ?F
  apply (simp only: wt-fun-env-def)
  apply (rule allI)+ apply (rule impI) apply (erule conjE)
proof -
  fix σ g t1 t2 e
  assume tf: (σ, g, t1, t2, e) ∈ ?F and ftv: FTV σ = {}

  have sft: (σ, Fun, g, TArrow t1 t2) ∈ ?T
  proof -
    from tf
    have (σ, g, t1, t2, e) = (τ, f, τ1', τ2', e')
      ∨ (σ, g, t1, t2, e) ∈ F1 apply auto done
    moreover { assume (σ, g, t1, t2, e) = (τ, f, τ1', τ2', e')
      hence (σ, Fun, g, TArrow t1 t2) ∈ ?T by simp
    } moreover { assume tf1: (σ, g, t1, t2, e) ∈ F1
      from tf1 ftv wtf have (σ, Fun, g, TArrow t1 t2) ∈ T1
      by (simp only: wt-fun-env-def, blast)
      hence (σ, Fun, g, TArrow t1 t2) ∈ ?T by auto
    } ultimately show (σ, Fun, g, TArrow t1 t2) ∈ ?T by blast
  
```

qed

have $e: (?T, \{\}, \text{Some } t1, e, \text{Some } t2) \in \text{wt-expr}$

proof –

from tf have $(\sigma, g, t1, t2, e) = (\tau, f, \tau1', \tau2', e')$
 $\vee (\sigma, g, t1, t2, e) \in F1$ **apply auto done**
moreover { **assume** $tp: (\sigma, g, t1, t2, e) = (\tau, f, \tau1', \tau2', e')$
 from $ftv\ tp\ epwt$ have $?T \mid \{\} \mid \text{Some } t1 \vdash e : \text{Some } t2$ **by simp**
} **moreover** { **assume** $tf1: (\sigma, g, t1, t2, e) \in F1$
 from $tf1\ ftv\ wtf$ have $T1 \mid \{\} \mid \text{Some } t1 \vdash e : \text{Some } t2$
 by (*simp only: wt-fun-env-def, blast*)
hence $?T \mid \{\} \mid \text{Some } t1 \vdash e : \text{Some } t2$
apply (*rule wt-expr-ext-tm*)
apply auto done
} **ultimately show** $?T \mid \{\} \mid \text{Some } t1 \vdash e : \text{Some } t2$ **by blast**

qed

have $tc: t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$

proof –

from tf have $(\sigma, g, t1, t2, e) = (\tau, f, \tau1', \tau2', e')$
 $\vee (\sigma, g, t1, t2, e) \in F1$ **apply auto done**
moreover { **assume** $tp: (\sigma, g, t1, t2, e) = (\tau, f, \tau1', \tau2', e')$

from $tp\ tv1\ ftv\ \text{eval-type-residual}$
have $t1r: t1 \in \text{residual-type}$ **by blast**
hence $t1c: FTV\ t1 = \{\}$ **by** (*rule residual-closed*)
 from $tp\ tv2\ ftv\ \text{eval-type-residual}$
have $t2r: t2 \in \text{residual-type}$ **by blast**
hence $t2c: FTV\ t2 = \{\}$ **by** (*rule residual-closed*)

from $tp\ t1c$ have $ft1p: FTV\ \tau1' = \{\}$ **by simp**
have $\tau1' \in (\text{ty-used-in } e' \cup \{\tau1', \tau2'\})$ **apply simp done**
with tp **have** $t1u: t1 \in (\text{ty-used-in } e' \cup \{\tau1', \tau2'\})$ **apply simp done**
from $ft1p\ t1u\ \text{inst}$ **have** $exm: \exists m. (\tau1', m) \in \text{lookup } T1$ **by auto**
from exm **obtain** $m1$ **where** $(\tau1', m1) \in \text{lookup } T1$ **apply blast done**
with tp **have** $t1l: (t1, m1) \in \text{lookup } T1$ **by simp**
have $\neg \text{tm-defined } t1\ T1 \vee (\text{tm-defined } t1\ T1)$ **by auto**
moreover { **assume** $ndt1: \neg \text{tm-defined } t1\ T1$
 from $t1u\ t1c\ t1l\ ndt1\ t2$ **have** $t1T2: (t1, m1) \in T2$
apply simp done
with $t1r$ **have** $t1 \in \text{compty } ?T$
apply (*simp only: compty-def*) **apply blast done**
} **moreover** { **assume** $tmd: \text{tm-defined } t1\ T1$
 from tmd **obtain** $p1\ m1'$ **where** $pt1: t1 \doteq p1$ **and** $pm1: (p1, m1') \in T1$
apply auto apply blast done
 from $t1r\ pt1\ \text{equiv-residual-eq}$ **have** $pt1eq: t1 = p1$
apply blast done
 with $pm1$ **have** $(t1, m1') \in T1$ **by simp**
 with $t1r$ **have** $t1 \in \text{compty } ?T$
apply (*simp only: compty-def*) **apply blast done**
} **ultimately have** $ct1: t1 \in \text{compty } ?T$ **by blast**

from $tp\ t2c$ have $ft2p: FTV\ \tau2' = \{\}$ **by simp**
have $\tau2' \in (\text{ty-used-in } e' \cup \{\tau1', \tau2'\})$ **apply simp done**
with tp **have** $t2u: t2 \in (\text{ty-used-in } e' \cup \{\tau1', \tau2'\})$ **apply simp done**
from $ft2p\ t2u\ \text{inst}$ **have** $exm2: \exists m. (\tau2', m) \in \text{lookup } T1$ **by auto**

```

from exm2 obtain m2 where  $(\tau 2', m2) \in \text{lookup } T1$  apply blast done
with tp have t2l:  $(t2, m2) \in \text{lookup } T1$  by simp
have  $\neg \text{tm-defined } t2 \ T1 \vee (\text{tm-defined } t2 \ T1)$  by auto
moreover { assume ndt2:  $\neg \text{tm-defined } t2 \ T1$ 
  from t2u t2c t2l ndt2 t2 have t2T2:  $(t2, m2) \in T2$  by simp
  with t2r have  $t2 \in \text{compty } ?T$ 
  apply (simp only: compty-def) apply blast done
} moreover { assume tmd:  $\text{tm-defined } t2 \ T1$ 
  from tmd obtain p1 m1' where pt1:  $t2 \doteq p1$  and pm1:  $(p1, m1') \in T1$ 
  apply auto apply blast done
  from t2r pt1 equiv-residual-eq have pt1eq:  $t2 = p1$ 
  apply blast done
  with pm1 have  $(t2, m1') \in T1$  by simp
  with t2r have  $t2 \in \text{compty } ?T$ 
  apply (simp only: compty-def) apply blast done
} ultimately have ct2:  $t2 \in \text{compty } ?T$  by blast
from ct1 ct2 have  $t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$  by blast
} moreover { assume tf1:  $(\sigma, g, t1, t2, e) \in F1$ 
  from tf1 ftv wtf have tc1:  $t1 \in \text{compty } T1 \wedge t2 \in \text{compty } T1$ 
  apply (simp only: wt-fun-env-def) apply blast done
  hence  $t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$  by auto
} ultimately show  $t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$  by blast
qed

```

```

from sft e tc show  $(\sigma, \text{Fun}, g, \text{TArrow } t1 \ t2) \in ?T \wedge$ 
   $(?T, \{\}, \text{Some } t1, e, \text{Some } t2) \in \text{wt-expr} \wedge$ 
   $t1 \in \text{compty } ?T \wedge t2 \in \text{compty } ?T$  apply blast done
qed

```

have *ndT*: *no-dups* *?T* **sorry**

have *nfT*: *no-freevars* *?T*

proof –

```

have nf2: no-freevars T2
  apply (simp only: no-freevars-def) apply clarify
proof –
  fix  $\pi \ k \ n \ \tau \ x$ 
  assume pt2:  $(\pi, k, n, \tau) \in T2$  and xft:  $x \in \text{FTV } \tau$ 
  from pt2 t2 have l:  $(\pi, k, n, \tau) \in \text{lookup } T1$  by simp
  from pt2 t2 have pf:  $\text{FTV } \pi = \{\}$  by simp
  from l pf nf have  $\text{FTV } \tau = \{\}$ 
  apply (rule lookup-freevars) done
  with xft have False by simp
  thus  $x \in \text{FTV } \pi$  by simp

```

qed

from *nf nf2* **have** *nf12*: *no-freevars* $(T1 \cup T2)$ **by** *auto*

```

from tv1 nf ty-eval-ftv have ft1:  $\text{FTV } \tau 1' \subseteq \{\}$  by blast
from tv2 nf ty-eval-ftv have ft2:  $\text{FTV } \tau 2' \subseteq \{\}$  by blast
from ft1 ft2 have  $\text{FTV } (\text{TArrow } \tau 1' \ \tau 2') \subseteq \{\}$  by simp
with nf12 show ?thesis apply simp done
qed

```

from *a fu wtenv ndT nfT ih*

show *fun-used-invar* $F' \ \{\}$ \wedge

wt-fun-env $T' \ F' \wedge \text{no-dups } T' \wedge \text{no-freevars } T'$

```

  apply blast done
qed
next
fix F T U  $\tau$  ans f tte
assume tff:  $(\tau, f, tte) \in F$ 
  and tfu:  $(\tau, f) \notin U$  and comp-prog T F U [] ans
  and ih: ?IH T F U ans
show ?IH T F (insert  $(\tau, f)$  U) ans
  apply (rule allI) apply (rule allI) apply (rule impI) apply (erule conjE)+
proof -
  fix T' F'
  assume a: ans = Some (T',F') and fu: fun-used-invar F (insert  $(\tau,f)$  U)
  and wtf: wt-fun-env T F and nd: no-dups T and nf: no-freevars T
  have fun-used-invar F U
  apply (simp only: fun-used-invar-def)
  apply (rule allI)+ apply (rule impI)
proof -
  fix  $\tau'$  f'
  assume tff2:  $(\tau', f') \in \text{fun-used-env } F$ 
  with fu have  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in \text{insert } (\tau, f) U$ 
  by simp
  moreover { assume x:  $\exists tte. (\tau', f', tte) \in F$ 
  from x obtain tte where  $(\tau', f', tte) \in F$  by auto
  hence  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in U$  by blast
} moreover { assume  $(\tau', f') \in \text{insert } (\tau, f) U$ 
  hence  $(\tau', f') = (\tau, f) \vee (\tau', f') \in U$  by auto
  moreover { assume  $(\tau', f') = (\tau, f)$ 
  with tff have  $(\tau', f', tte) \in F$  by simp
  hence  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in U$  by blast
} moreover { assume  $(\tau', f') \in U$ 
  hence  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in U$  by auto
} ultimately have  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in U$  by auto
} ultimately show  $(\exists tte. (\tau', f', tte) \in F) \vee (\tau', f') \in U$  by auto
qed
with a wtf nd nf ih show fun-used-invar F' {}  $\wedge$  wt-fun-env T' F'
 $\wedge$  no-dups T'  $\wedge$  no-freevars T' apply blast done
qed
qed

```

theorem compile-sound:

```

  assumes cp: {} | {} | {}  $\models p \Downarrow$  Some (T',F')
  and mainT: (TmId 0 TInt TInt, Fun, 0, TArrow TInt TInt)  $\in T'$ 
  and mainF: (TmId 0 TInt TInt, 0, TInt, TInt, e)  $\in F'$ 
  and run: F' | T'  $\vdash$  EApp (EMem (TmId 0 TInt TInt) 0) (EInt 0)  $\Rightarrow a$ 
shows  $\exists v. a = \text{Some } v \wedge T' | \{\} | \text{None} \vdash v : \text{Some } TInt$ 
proof -
  let ?E = EApp (EMem (TmId 0 TInt TInt) 0) (EInt 0)
  have wt: T' | {} | None  $\vdash$  ?E : Some TInt
  proof -
    from mainT have 0  $\in$  tms T' apply (simp add: tms-def) apply blast done
    hence wf: T' | {}  $\vdash$  TmId 0 TInt TInt wf apply auto done
    from wf mainT
    have T' | {} | None  $\vdash$  EMem (TmId 0 TInt TInt) 0 : Some (TArrow TInt TInt)
    apply blast done
  thus ?thesis by auto
qed

```

```

have fi: fun-used-invar {} {} by auto
have wte: wt-fun-env {} {} by auto
have nd: no-dups {} by auto
have nf: no-freevars {} by auto

from cp fi wte nd nf compile-preserves
have A: fun-used-invar  $F'$  {}  $\wedge$  wt-fun-env  $T' F' \wedge$  no-dups  $T' \wedge$  no-freevars  $T'$ 
  apply blast done

have fud: fun-used-defd  $F'$  (fun-used-in ? $E \cup$  fun-used-env  $F'$ )
  apply (simp only: fun-used-defd-def) apply clarify
proof -
  fix  $\tau f$ 
  assume  $(\tau, f) \in$  fun-used-in ? $E \cup$  fun-used-env  $F'$ 
  moreover { assume  $(\tau, f) \in$  fun-used-in ? $E$ 
    hence  $(\tau, f) = (TmId\ 0\ TInt\ TInt, 0)$  apply auto done
    with mainF have  $\exists tte. (\tau, f, tte) \in F'$  by auto
  } moreover { assume  $(\tau, f) \in$  fun-used-env  $F'$ 
    with A have  $\exists tte. (\tau, f, tte) \in F'$  apply auto done
  } ultimately show  $\exists tte. (\tau, f, tte) \in F'$  apply blast done
qed
from A have wtenv: wt-fun-env  $T' F'$  by blast
from A have nd: no-dups  $T'$  by blast

from wt fud wtenv nd run wt-expr-sound
show  $\exists v. a = Some\ v \wedge T' \mid \{\} \mid None \vdash v : Some\ TInt$ 
  apply blast done
qed

end

```