CS 336	**Programming Languages**	**Homework Solution 4**
Winter 2005		**Due 2/24/05**

1 Problem 1. (20 pts)

Do Exercise 18.6.2.

We define a *meta*-operation $+$ on types as follows: If R is a record type with labels given by $labels(\mathsf{R})$ and with the field type for label m denoted by $\mathsf{R}(m)$, then

$$\mathsf{R}' = \mathsf{R} + \{\mathsf{l} : \mathsf{T}_1\}$$

is a record type such that

$$labels(\mathsf{R}') = labels(\mathsf{R}) \cup \{\mathsf{l}\}$$

and with field types given by

$$\begin{aligned} \mathsf{R}'(m) &= \mathsf{T}_1 \text{ if } m = \mathsf{l} \\ &= \mathsf{R}(m) \text{ if } m \in labels(\mathsf{R}) - \{l\} \end{aligned}$$

That is $\mathsf{R} + \{l : \mathsf{T}_1\}$ is a record that *extends* R with a new field with label l of type T_1. The label l may be among the fields of R, in which case the existing field is overridden. Note that the $+$ operator is not part of the type language; it is a meta-notation for expressing a derived record type.

This version of the `with` operation assumes that we are adding/overriding one field. A more general version would allow the the concatenation of two arbitrary record values. This is a fairly straightforward generalization of the development below.

Syntax. The only syntactic category that is changed is terms, and types and values remain as before:

$$\mathsf{t} ::= \ldots \mid \mathsf{t} \text{ with } \{\mathsf{l} : \mathsf{T}\}$$

Typing Rules. There is one new typing rule:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{R} \qquad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{t}_1 \text{ with}\{\mathsf{l} : \mathsf{t}_2\} : \mathsf{R} + \{\mathsf{l} : \mathsf{T}_2\}} \qquad \text{(T-WITH)}$$

Evaluation Rules.

$$\frac{\mathsf{t}_1 \longrightarrow \mathsf{t}_1'}{\mathsf{t}_1 \text{ with } \{\mathsf{l} : \mathsf{t}_2\} \longrightarrow \mathsf{t}_1' \text{ with } \{\mathsf{l} : \mathsf{t}_2\}} \qquad \text{(E-WITHLEFT)}$$

$$\frac{\mathsf{t}_2 \longrightarrow \mathsf{t}_2'}{\mathsf{v}_1 \text{ with } \{\mathsf{l} : \mathsf{t}_2\} \longrightarrow \mathsf{v}_1 \text{ with } \{\mathsf{l} : \mathsf{t}_2'\}} \qquad \text{(E-WITHRIGHT)}$$

$$\mathsf{v}_1 \text{ with } \{\mathsf{l} : \mathsf{v}_2\} \longrightarrow \mathsf{v} \qquad \text{(E-WITH)}$$

where v is the concatenation of the record values:

$$labels(\mathsf{v}) = labels(\mathsf{v}_1) \cup \{\mathsf{l}\}$$

$$\begin{array}{rcl} \mathsf{v}(m) & = & \mathsf{v}_2 \quad \text{if } m = \mathsf{l} \\ & = & \mathsf{v}_1(m) \quad \text{if } m \in labels(\mathsf{v}_1) - \{\mathsf{l}\} \end{array}$$

2 Problem 2. (20 pts)

Redo the examples of Section 20.1 using the isorecursive types of Figure 20.1.

Lists.

$$\mathsf{NatList} = \mu \mathsf{X}.\langle \mathsf{nil} : \mathsf{Unit}, \mathsf{cons} : \mathsf{Nat} \times \mathsf{X} \rangle$$

$$\mathsf{nil} = \mathsf{fold}[\mathsf{NatList}](\langle \mathsf{nil} = \mathsf{unit} \rangle \text{ as } \langle \mathsf{nil} : \mathsf{Unit}, \mathsf{cons} : \mathsf{Nat} \times \mathsf{NatList} \rangle)$$

$$\mathsf{cons} = \lambda \mathsf{n} : \mathsf{Nat}.\lambda \mathsf{l} : \mathsf{NatList}.(\langle \mathsf{cons} = (\mathsf{n}, \mathsf{l}) \rangle \text{ as } \langle \mathsf{nil} : \mathsf{Unit}, \mathsf{cons} : \mathsf{Nat} \times \mathsf{NatList} \rangle)$$

$$\mathsf{isnil} = \lambda \mathsf{l} : \mathsf{NatList}.\ \mathsf{case}\ \mathsf{unfold}[\mathsf{NatList}]\ \mathsf{l}\ \mathsf{of}\ \langle \mathsf{nil} = \mathsf{u} \rangle \Rightarrow \mathsf{true} | \langle \mathsf{cons} = \mathsf{p} \rangle \Rightarrow \mathsf{false}$$

$$\mathsf{hd} = \lambda \mathsf{l} : \mathsf{NatList}.\ \mathsf{case}\ \mathsf{unfold}[\mathsf{NatList}]\ \mathsf{l}\ \mathsf{of}\ \langle \mathsf{nil} = \mathsf{u} \rangle \Rightarrow 0 | \langle \mathsf{cons} = \mathsf{p} \rangle \Rightarrow \mathsf{p}.1$$

$$\mathsf{tl} = \lambda \mathsf{l} : \mathsf{NatList}.\ \mathsf{case}\ \mathsf{unfold}[\mathsf{NatList}]\ \mathsf{l}\ \mathsf{of}\ \langle \mathsf{nil} = \mathsf{u} \rangle \Rightarrow \mathsf{nil} | \langle \mathsf{cons} = \mathsf{p} \rangle \Rightarrow \mathsf{p}.2$$

$$\mathsf{sumlist} = unchanged$$

Hungry

$$\mathsf{Hungry} = \mu \mathsf{A}.\mathsf{Nat} \to \mathsf{A}$$

$$\mathsf{f} = \mathsf{fix}(\lambda \mathsf{f} : \mathsf{Nat} \to \mathsf{Hungry}.\ \lambda \mathsf{n} : \mathsf{Nat}.\ \mathsf{fold}[\mathsf{Hungry}]\ f$$

Streams

$$\mathsf{Stream} = \mu \mathsf{A}.\ \mathsf{Unit} \to \mathsf{Nat} \times \mathsf{A}$$

$$\mathsf{hd} = \lambda \mathsf{s}.\ (\mathsf{unfold}[\mathsf{Stream}]\ s\ \mathsf{unit}).1$$

$$\mathsf{tl} = \lambda \mathsf{s}.\ (\mathsf{unfold}[\mathsf{Stream}]\ s\ \mathsf{unit}).2$$

$$\mathsf{upfrom} = \mathsf{fix}(\lambda \mathsf{f} : \mathsf{Nat} \to \mathsf{Stream}.\ \lambda \mathsf{n} : \mathsf{Nat}.\ \mathsf{fold}[\mathsf{Stream}]\ (\lambda_{\_} : \mathsf{Unit}.\ (\mathsf{n}, \mathsf{f}(\mathsf{succ}(\mathsf{n})))))$$

$$\mathsf{upfrom0} = \mathsf{upfrom}\ 0$$

Processes

$$\mathsf{Process} \;=\; \mu \mathsf{A}.\, \mathsf{Nat} \to (\mathsf{Nat} \times \mathsf{A})$$

$$\mathsf{p1} \;=\; \mathsf{fix}(\lambda \mathsf{f} : \mathsf{Nat} \to \mathsf{Process}.\, \lambda \mathsf{acc} : \mathsf{Nat}.\, \mathsf{fold}[\mathsf{Process}](\lambda \mathsf{n} : \mathsf{Nat}.\, \mathsf{let\ nacc} = \mathsf{plus\ acc\ n\ in\ } (\mathsf{nacc}, \mathsf{f\ nacc})))$$

$$\mathsf{p} \;=\; \mathsf{p}\, 0$$

$$\mathsf{curr} \;=\; \lambda \mathsf{s} : \mathsf{Process}.\, (\mathsf{unfold}[\mathsf{Process}]\; s\; 0).1$$

$$\mathsf{send} \;=\; \lambda \mathsf{n}.\, \lambda \mathsf{s} : \mathsf{Process}.\, (\mathsf{unfold}[\mathsf{Process}]\; s\; \mathsf{n}).2$$

Objects

$$\mathsf{Counter} \;=\; \mu \mathsf{C}.\, \{\mathsf{get} : \mathsf{Nat},\ \mathsf{inc} : \mathsf{Unit} \to \mathsf{C},\ \mathsf{dec} : \mathsf{Unit} \to \mathsf{C}\}$$

$$\begin{aligned}
\mathsf{c} \;=\; & \mathsf{let\ create} = \mathsf{fix}(\lambda \mathsf{f} : \{\mathsf{x} : \mathsf{Nat}\} \to \mathsf{Counter}.\, \lambda \mathsf{s} : \{\mathsf{x} : \mathsf{Nat}\}. \\
& \qquad \mathsf{fold}[\mathsf{Counter}] \\
& \qquad\quad \{\mathsf{get} = \mathsf{s.x}, \\
& \qquad\quad\ \mathsf{inc} = \lambda_ : \mathsf{Unit}.\, \mathsf{f}\{\mathsf{x} = \mathsf{succ}(ms.x)\}, \\
& \qquad\quad\ \mathsf{dec} = \lambda_ : \mathsf{Unit}.\, \mathsf{f}\{\mathsf{x} = \mathsf{pred}(ms.x)\}\}) \\
& \mathsf{in\ create}\ \{\mathsf{x} = 0\}
\end{aligned}$$

Fixed Point Operator (CBN)

$$\begin{aligned}
\mathsf{fix}_T \;=\; & \lambda \mathsf{f} : \mathsf{T} \to \mathsf{T}. \\
& \quad (\lambda \mathsf{x} : (\mu \mathsf{A}.\mathsf{A} \to \mathsf{T}).\, \mathsf{f}(\mathsf{unfold}[\mu \mathsf{A}.\mathsf{A} \to \mathsf{T}]\, \mathsf{x}\, \mathsf{x})) \\
& \quad (\mathsf{fold}[\mu \mathsf{A}.\mathsf{A} \to \mathsf{T}]\, ((\lambda \mathsf{x} : (\mu \mathsf{A}.\mathsf{A} \to \mathsf{T}).\, \mathsf{f}(\mathsf{unfold}[\mu \mathsf{A}.\mathsf{A} \to \mathsf{T}]\, \mathsf{x}\, \mathsf{x})))
\end{aligned}$$

Untyped Lambda Calculus

$$\mathsf{D} \;=\; \mu \mathsf{X}.\, \mathsf{X} \to \mathsf{X}$$

$$\mathsf{lam} \;=\; \lambda \mathsf{f} : \mathsf{D} \to \mathsf{D}.\, \mathsf{fold}[\mathsf{D}]\, \mathsf{f}$$

$$\mathsf{ap} \;=\; \lambda \mathsf{f} : \mathsf{D}.\, \lambda \mathsf{a} : \mathsf{D}.\, \mathsf{unfold}[\mathsf{D}]\, \mathsf{f}\, \mathsf{a}$$

$$\mathsf{D} \;=\; \mu \mathsf{X}.\, \langle \mathsf{nat} : \mathsf{Nat}, \mathsf{fn} : \mathsf{X} \to \mathsf{X} \rangle$$

$$\mathsf{lam} \;=\; \lambda \mathsf{f} : \mathsf{D} \to \mathsf{D}.\, \mathsf{fold}[\mathsf{D}]\, (\langle \mathsf{fn} = \mathsf{f} \rangle\ \mathsf{as}\ \langle \mathsf{nat} : \mathsf{Nat}, \mathsf{fn} : \mathsf{D} \to \mathsf{D} \rangle)$$

$$\mathsf{ap} \;=\; \lambda \mathsf{f} : \mathsf{D}.\, \lambda \mathsf{a} : \mathsf{D}.\, \mathsf{case\ unfold}[\mathsf{D}]\, \mathsf{f}\ \mathsf{of}\ \langle \mathsf{nat} = \mathsf{n} \rangle \Rightarrow \mathsf{diverge}_{\mathsf{D}}\ \mathsf{unit} \mid \langle \mathsf{fn} = \mathsf{f} \rangle \Rightarrow \mathsf{f\ aa}$$

3 Problem 3. (20 pts)

Redo the examples of the previous exercise in Standard ML.

NatList

```
datatype NatList = NIL | CONS of int * NatList

val cons = (fn (n,l) => cons (n,l))

fun isnil NIL = true
  | isnil _ = false

fun hd NIL = 0
  | hd (CONS(n,_)) = n

fun tl NIL = NIL
  | tl (CONS(_,l)) = l

fun sumlist (l: NatList) =
    if isnil l then 0 else hd l + sumlist (tl l)

fun sumlist NIL = 0
  | sumlist (n::l) = n + sumlist l
```

Hungry

```
datatype Hungry = H of int -> Hungry

fun f0 n = H f0

val f : Hungry = H f0

fun ap (H f: Hungry) n = f n

ap (ap f 0) 1
```

Stream

```
datatype Stream = S of unit -> int * Stream

fun hd (S f) = #1(f ())

fun tl (S f) = #2(f ())

fun upfrom n = S(fn () => (n, upfrom(n + 1)))

val upfrom0 = upfrom 0
```

Process

```
datatype Process = P of int -> (int * Process)

fun pf acc = P(fn n => let val newacc = acc + n in (newacc, pf newacc) end

val p = pf 0

fun curr (P s: Process) = #1(s 0)

fun send (n: int) (P s: Process) = #2(s n)
```

Fixed Point Operator (CBV)

```
datatype 'a f = F of 'a f -> 'a

fun unF (F f) = f

fun 'a fix (f : 'a -> 'a) =
    (fn (x: 'a f) => f ((unF x) x))(F(fn (x: 'a f) => f ((unF x) x)))
```

Untyped Lambda Calculus

```
datatype D = MkD of D -> D

fun lam (f : D -> D) = MkD f

fun ap (MkD f: D) (a: D) = f a

datatype D' = Nat of int | Fn of D' -> D'

fun lam' (f: D' -> D') = Fn f

fun ap' (Fn f: D') (a: D') = f a
  | ap' (Nat n: D') (a: D') = raise Fail "ap'"
```

4 Problem 4. (20 pts)

Prove Theorem 23.5.1 (Preservation for the polymorphic lambda calculus, Figure 23.1). Give only the new cases involving the polymorphic constructs of the language.

Theorem: $\Gamma \vdash t : T \ \& \ t \to t' \ \Rightarrow \ \Gamma \vdash t' : T$.

Proof: We prove this by induction on the rules deriving $t \to t'$. We need only deal with the new cases involving polymorphism, namely the evaluation rules (E-TAPP) and (E-TAPPTABS).

Case: $t \to t'$ by (E-TAPP).
So $t = t_1[T_2]$ and $t' = t_1'[T_2]$ where:

$$(1) \qquad t_1 \to t_1'$$

By Inversion, there exists T_{12} such that

$$(2) \qquad T = [X \mapsto T_2]T_{12}$$

$$(3) \qquad \Gamma \vdash t_1 : \forall X.T_{12}$$

Then by the Induction Hypothesis and (1) and (2) we have

$$(4) \qquad \Gamma \vdash \mathsf{t}_1' : \forall \mathsf{X}.\mathsf{T}_{12}$$

Therefore by (T-TAPP) we have:

$$(5) \qquad \Gamma \vdash \mathsf{t}_1'[\mathsf{T}_2] : [\mathsf{X} \mapsto \mathsf{T}_2]\mathsf{T}_{12}$$

and hence

$$(6) \qquad \Gamma \vdash \mathsf{t}' : \mathsf{T}$$

Case: $\mathsf{t} \to \mathsf{t}'$ by (E-TAPPTABS).
Then for some X, t_{12}, T_2 we have

$$(1) \qquad \mathsf{t} = (\lambda \mathsf{X}.\, \mathsf{t}_{12})[\mathsf{T}_2]$$

$$(2) \qquad \mathsf{t}' = [\mathsf{X} \mapsto \mathsf{T}_2]\, \mathsf{t}_{12}$$

By inversion of $\Gamma \vdash (\lambda \mathsf{X}.\, \mathsf{t}_{12})[\mathsf{T}_2] : \mathsf{T}$ there exists T_{12} such that

$$(3) \qquad \mathsf{T} = [\mathsf{X} \mapsto \mathsf{T}_2]\, \mathsf{T}_{12}$$

$$(4) \qquad \Gamma \vdash \lambda \mathsf{X}.\, \mathsf{t}_{12} : \forall \mathsf{X}.\, \mathsf{T}_{12}$$

Then by Inversion of (4) we have

$$(5) \qquad \Gamma, \mathsf{X} \vdash \mathsf{t}_{12} : \mathsf{T}_{12}$$

Now we need to make use of the following Substitution Lemma for types:

Lemma[Substitution for Types]. For any S,

$$\Gamma, \mathsf{X}, \Delta \vdash \mathsf{t} : \mathsf{T} \;\Rightarrow\; \Gamma, [\mathsf{X} \mapsto \mathsf{S}]\Delta \vdash [\mathsf{X} \mapsto \mathsf{S}]\mathsf{t} : [\mathsf{X} \mapsto \mathsf{S}]\mathsf{T}$$

Now applying this lemma to (5) with $\mathsf{S} = \mathsf{T}_2$ and $\Delta = \emptyset$, we have

$$(6) \qquad \Gamma \vdash [\mathsf{X} \mapsto \mathsf{T}_2]\mathsf{t}_{12} : [\mathsf{X} \mapsto \mathsf{T}_2]\mathsf{T}_{12}$$

and hence, by (2) and (3),

$$(7) \qquad \Gamma \vdash \mathsf{t}' : \mathsf{T}$$

Proof of Substitution Lemma for Types.

We prove this by induction on the typing rules.

For any construct α, let α^* be $[\mathsf{X} \mapsto \mathsf{S}]\alpha$. So we must show that

$$(1) \qquad \Gamma, \Delta^* \vdash \mathsf{t}^* : \mathsf{T}*$$

Case: The hypothesis holds by (T-VAR).
Then $\mathsf{t} = \mathsf{x}$ and by Inversion, $\mathsf{x} : \mathsf{T} \in \Gamma, \mathsf{X}, \Delta$. Note also that $x^* = x$.

There are two cases:
(i) $\mathsf{x} : \mathsf{T} \in \Gamma$: Then we note that X is not free in Γ, and therefore $\mathsf{T}^* = \mathsf{T}$. So

$$(2) \qquad \Gamma, \vdash \mathsf{x} : \mathsf{T}$$

which is equivalent to

$$(3) \qquad \Gamma, \vdash \mathsf{x}^* : \mathsf{T}^*$$

Then by the appropriate Weakening Lemma, noting that $\mathsf{x} \notin Dom(\Delta)$, we have:

$$(4) \qquad \Gamma, \Delta^* \vdash \mathsf{x}^* : \mathsf{T}^*$$

(ii) $\mathsf{x} : \mathsf{T} \in \Delta$: In this case, X may occur free in T. It is clear from the definition of $[\mathsf{X} \mapsto \mathsf{S}]\Delta$ that

$$(5) \qquad \mathsf{x} : [\mathsf{X} \mapsto \mathsf{S}]\mathsf{T} \in [\mathsf{X} \mapsto \mathsf{S}]\Delta$$

$$(6) \qquad [\mathsf{X} \mapsto \mathsf{S}]\Delta \vdash \mathsf{x} : [\mathsf{X} \mapsto \mathsf{S}]\mathsf{T} \quad \text{by (T-VAR)}$$

and hence by Weakening

$$(7) \qquad \Gamma, [\mathsf{X} \mapsto \mathsf{S}]\Delta \vdash \mathsf{x} : [\mathsf{X} \mapsto \mathsf{S}]\mathsf{T} \quad \text{QED}$$

Case: (T-ABS) so $\mathsf{t} = \lambda\mathsf{x} : \mathsf{T}_1.\mathsf{t}_2$ and $\mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2$, where

$$(8) \qquad \Gamma, \mathsf{X}, \Delta, \mathsf{x} : \mathsf{T}_1 \vdash \mathsf{t}_2 : \mathsf{T}_2$$

By the Induction Hypothesis,

$$(9) \qquad \Gamma, \mathsf{X}, (\Delta, \mathsf{x} : \mathsf{T}_1)^* \vdash \mathsf{t}_2^* : \mathsf{T}_2^* \quad \text{hence}$$

$$(10) \qquad \Gamma, \mathsf{X}, \Delta^*, \mathsf{x} : \mathsf{T}_1^* \vdash \mathsf{t}_2^* : \mathsf{T}_2^* \quad \text{hence, by (T-ABS)}$$

$$(11) \qquad \Gamma, \mathsf{X}, \Delta^* \vdash (\lambda\mathsf{x} : \mathsf{T}_1^*.\, \mathsf{t}_2^*) : \mathsf{T}_2^* \quad \text{hence}$$

$$(12) \qquad \Gamma, \mathsf{X}, \Delta^* \vdash (\lambda\mathsf{x} : \mathsf{T}_1.\, \mathsf{t}_2)^* : \mathsf{T}_2^* \quad \text{QED}$$

Case: (T=TABS) so $\mathsf{t} = \lambda\mathsf{Y}.\mathsf{t}_1$ and $\mathsf{T} = \forall\mathsf{Y}.\mathsf{T}_1$, where

$$(13) \qquad \Gamma, \mathsf{X}, \Delta, \mathsf{Y} \vdash \mathsf{t}_1 : \mathsf{T}_1$$

and we can assume $\mathsf{X} \neq \mathsf{Y}$ and hence $\mathsf{Y}^* = \mathsf{Y}$. By the Induction Hypothesis

$$(14) \qquad \Gamma, \mathsf{X}, (\Delta, \mathsf{Y})^* \vdash \mathsf{t}_1^* : \mathsf{T}_1^*, \quad \text{or}$$

$$(15) \qquad \Gamma, \mathsf{X}, \Delta^*, \mathsf{Y}^* \vdash \mathsf{t}_1^* : \mathsf{T}_1^*, \quad \text{hence}$$

$$(16) \qquad \Gamma, \mathsf{X}, \Delta^*, \mathsf{Y} \vdash \mathsf{t}_1^* : \mathsf{T}_1^*, \quad \text{since } \mathsf{Y}^* = \mathsf{Y}$$

Then by (T-TABS) we have

$$(17) \qquad \Gamma, \mathsf{X}, \Delta^* \vdash \lambda\mathsf{Y}.\, \mathsf{t}_1^* : \forall\mathsf{Y}.\, \mathsf{T}_1^*, \quad \text{or, equivalently,}$$

$$(18) \qquad \Gamma, \mathsf{X}, \Delta^* \vdash (\lambda\mathsf{Y}.\, \mathsf{t}_1)^* : (\forall\mathsf{Y}.\, \mathsf{T}_1)^*, \quad \text{QED}$$

The other, simpler cases are left as exercises.

5 Problem 5. (20 pts)

Prove the Progress theorem for Existential types (Figure 24.1). Give only the new cases involving the existential type constructs.

Theorem: $\vdash \mathsf{t} : \mathsf{T} \Rightarrow \mathsf{t}$ is a value, or $\exists\mathsf{t}'.\, \mathsf{t} \to \mathsf{t}'$

Proof: We prove this by induction on the typing rules for $\vdash \mathsf{t} : \mathsf{T}$, doing only the cases associated with existential types.

Case: (T-PACK). So

$$(1) \qquad \mathsf{t} = \{\mathsf{U}, \mathsf{t}_2\} \text{ as } \{\exists\mathsf{X},\ \mathsf{T}_2\}$$

(2) $\mathsf{T} = \{\exists \mathsf{X},\ \mathsf{T}_2\}$

By Inversion, we have

(3) $\Gamma \vdash \mathsf{t}_2 : [\mathsf{X} \mapsto \mathsf{U}]\mathsf{T}_2$

By the Induction Hypothesis, either

(4) t_2 is a value, or

(5) $\exists \mathsf{t}_2'.\, \mathsf{t}_2 \longrightarrow \mathsf{t}_2'$

If t_2 is a value, then so is t, and we are done. So suppose that (5) holds. Then by (E-PACK), we have

(6) $\mathsf{t} \longrightarrow \{\mathsf{U}, \mathsf{t}_2'\}\ \mathsf{as}\ \{\exists \mathsf{X},\ \mathsf{T}_2\}$ QED

Case: (T-UNPACK). So

(1) $\mathsf{t} = \mathsf{let}\ \{\mathsf{X}, \mathsf{x}\} = \mathsf{t}_1\ \mathsf{in}\ \mathsf{t}_2$

By Inversion, there exists a type T_{12} such that

(2) $\Gamma \vdash \mathsf{t}_1 : \{\exists \mathsf{X},\ \mathsf{T}_{12}\}$ and

(3) $\Gamma, \mathsf{X}, \mathsf{x} : \mathsf{T}_{12} \vdash \mathsf{t}_2 : \mathsf{T}$

By the Induction Hypothesis and 2), we have either

(4) t_1 is a value, or

(5) $\exists \mathsf{t}_1'.\, \mathsf{t}_1 \longrightarrow \mathsf{t}_1'$

If (4) is the case, then by the Canonical Forms Lemma (appropriately extended), we have

(6) $\mathsf{t}_1 = \{{}^*\mathsf{U}, \mathsf{v}_1\} as \{\exists \mathsf{X},\ \mathsf{T}_{12}\}$

for some type U and value v_1. Then by (E-UNPACKPACK) we have

(7) $\mathsf{t} \longrightarrow [\mathsf{X} \mapsto \mathsf{U}][\mathsf{x} \mapsto \mathsf{v}_1]\mathsf{t}_2$

and we are done. If (5) holds, then by (E-UNPACK), we have

(8) $\mathsf{t} \longrightarrow \mathsf{let}\ \{\mathsf{X}, \mathsf{x}\} = \mathsf{t}_1'\ \mathsf{in}\ \mathsf{t}_2$

and so t makes a transition and we are done.