CIS 500 — Software Foundations

Midterm II

Answer key

November 13, 2002

Simply typed lambda-calculus

The definition of the simply typed lambda-calculus with `Unit` is reproduced on page 11.

1. (8 points) For each of the following untyped λ-terms, either give a well-typed term of the simply typed lambda-calculus with `Unit` whose erasure is the given term, or else write "not typable" if no such term exists.

   The type annotations in your answers should only involve `Unit` and →.

   (a) `λx. x (x unit)`
   Answer: *`λx:Unit→Unit. x (x unit)`*

   (b) `λx. x unit x`
   Answer: Not typable

   (c) `λx. x unit unit`
   Answer: *`λx:Unit→Unit→Unit. x unit unit`*
   (for example)

   (d) `λx. λy. λz. (x y) (y z)`
   Answer: *`λx:(Unit→Unit)→Unit→Unit. λy:Unit→Unit. λz:Unit. (x y) (y z)`*
   (for example)

   Grading scheme: Each of the items is worth 2 points. Partial credit (1 point) was given for incorrect but "not completely unreasonable" answers.

References

The definition of the simply typed lambda-calculus with references is reproduced on page 11.

2. (9 points) Suppose, for this question, that our language also has `let` expressions and numbers. Then evaluating the expression

```
let x = ref 5 in
let y = x in
let z = ref (λa:Nat. y := a; succ (!y)) in
(!z) (!y)
```

beginning in an empty store yields:

Result: 6 Store: $l_1 \mapsto$ 5
$l_2 \mapsto$ `λa:Nat.` l_1 `:= a; succ (!`l_1`)`

Fill in the results and final stores (when started with an empty store) of the following terms:

(a)
```
let x = ref 2 in
let y = x in
let f = λa:Ref Nat. λb:Ref Nat. a := 5; b := 6; !a in
f x y
```
Answer: Result: 6 Store: $l_1 \mapsto$ 6

(b)
```
let x = ref 2 in
let y = ref x in
let z = ref y in
!z
```
Answer: Result: l_2 Store: $l_1 \mapsto$ 2
$l_2 \mapsto l_1$
$l_3 \mapsto l_2$

(c)
```
let x = ref 0 in
let f = ref (λu:Unit. !x) in
x := 2;
let g = λu:Unit. (!f) unit in
x := 3;
f := λu:Unit. succ (!x) in
let r = g unit in
x := 9;
r
```
Answer: Result: 4 Store: $l_1 \mapsto$ 9
$l_2 \mapsto$ `λu:Unit. succ (!`l_1`)`

Grading scheme: Each of the items was worth 3 points (1 point for the correct result, and 2 points for the correct store). Partial credit was given for "somewhat correct" stores; for instance,

$l_2 \mapsto$ *`λu:Unit. succ (!x)`*

was accepted as an almost correct variant of

$l_2 \mapsto$ *`λu:Unit. succ (!`l_1`)`.*

3. (3 points) Is there any well-typed term that, when started with an empty store, will yield the following store?

   $l_1 \mapsto l_1$

   If so, give one. If not, explain (briefly!) why not.

   Answer: No: this store is not typable, so the preservation theorem tells us that no well-typed program could create it.

   Grading scheme:

   - *1 point for "no" (0 points for "yes")*
   - *2 points for saying something about why it seems hard to create, but omitting the observation about typing.*

4. (8 points) We saw in homework 8 that, using references, we can achieve the effect of a recursive function definition by building a “cyclic store” in which the function’s body refers to its own definition indirectly, via a reference cell. The same idea extends straightforwardly to mutually recursive definitions.

   Fill in the blanks in the following expressions so that, after evaluating them, `even` will be a function that checks whether its argument `n` is even (by returning `true` if it is `0` and otherwise checking whether `(pred n)` is odd).

```
even_ref = ref (λn:Nat.true);
odd_ref = ref (λn:Nat.true);

even_body = λn:Nat. if iszero n then true else ((__________)(pred n));

odd_body = λn:Nat. if iszero n then false else ((__________)(pred n));

even_ref := __________________;

odd_ref  := __________________;

even = !even_ref;
odd = !odd_ref;
```

   Answer:

```
even_body = λn:Nat. if iszero n then true else ((!odd_ref)(pred n));
odd_body = λn:Nat. if iszero n then false else ((!even_ref)(pred n));
even_ref := even_body;
odd_ref := odd_body;
```

   Grading scheme: The problem asked to simulate *recursion through references. However, some people defined* even$_{body}$ *using* odd$_{body}$*, i.e., through the regular recursion. If this did not contain additional errors, 2 points; zero otherwise.*

   If the dereference operator ! was missing in the entries for even$_{body}$ *and* odd$_{body}$*, then minus 4 points. If the extraneous* ref *operator was present in the assignements for* even$_{ref}$ *and* odd$_{ref}$*, then minus 4 points.*

   If the suffixes ref *or* body *were absent in the function names, minus 2 points.*

   Minus 2 points for an error in the function logic, e.g. odd *instead of* even*, or an extra negation of a function call, etc.*

5. (20 points) In Chapter 13 of TAPL, the following lemmas were used in proving the preservation property for the simply typed lambda-calculus with references. (We've given all the lemmas names here, for easy reference.)

LEMMA [INVERSION]:

(a) If $\Gamma \mid \Sigma \vdash \mathtt{x} : \mathtt{T}$, then $\mathtt{x}{:}\mathtt{T} \in \Gamma$.
(b) If $\Gamma \mid \Sigma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2 : \mathtt{T}$, then $\mathtt{T} = \mathtt{T}_1 \to \mathtt{T}_2$ for some $\mathtt{T}_2$ with $\Gamma, \mathtt{x}{:}\mathtt{T}_1 \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2$.
(c) If $\Gamma \mid \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}$, then there is some type $\mathtt{T}_{11}$ such that $\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$.
(d) If $\Gamma \mid \Sigma \vdash \mathtt{unit} : \mathtt{T}$, then $\mathtt{T} = \mathtt{Unit}$.
(e) If $\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{T}$, then $\mathtt{T} = \mathtt{Ref}\ \mathtt{T}_1$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathtt{T}_1$.
(f) If $\Gamma \mid \Sigma \vdash\ !\mathtt{t}_1 : \mathtt{T}$, then $\mathtt{T} = \mathtt{T}_{11}$ with $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathtt{Ref}\ \mathtt{T}_{11}$.
(g) If $\Gamma \mid \Sigma \vdash \mathtt{t}_1 := \mathtt{t}_2 : \mathtt{T}$, then $\mathtt{T} = \mathtt{Unit}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathtt{Ref}\ \mathtt{T}_{11}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$.
(h) If $\Gamma \mid \Sigma \vdash l : \mathtt{T}$, then $\mathtt{T} = \mathtt{Ref}\ \Sigma(l)$.

LEMMA [SUBSTITUTION]: If $\Gamma, \mathtt{x}{:}\mathtt{S} \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \mid \Sigma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.

LEMMA [REPLACEMENT]: If

$\Gamma \mid \Sigma \vdash \mu$
$\Sigma(l) = \mathtt{T}$
$\Gamma \mid \Sigma \vdash \mathtt{v} : \mathtt{T}$

then $\Gamma \mid \Sigma \vdash [l \mapsto \mathtt{v}]\mu$.

LEMMA [WEAKENING]: If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Sigma' \supseteq \Sigma$, then $\Gamma \mid \Sigma' \vdash \mathtt{t} : \mathtt{T}$.

For each case in the proof on the next page, write down the *skeleton* of the argument. A skeleton contains the same sequence of steps as the full argument, but omits all details. The rules for writing skeletons are as follows:

- Steps of the form "By part (x) of the inversion lemma, we obtain..." in the full argument become "inversion(x)" in the skeleton.
- Steps of the form "By the substitution lemma, we obtain..." become "substitution." (Similarly for replacement and weakening.)
- Steps of the form "By the induction hypothesis, we obtain..." become "IH."
- Steps of the form "By typing rule T-XXX, we obtain..." become "T-XXX."
- If the full argument doesn't use any of the lemmas or the induction hypothesis, then its skeleton is "Direct."

For example, if the full argument is

Case E-DEREFLOC*:* $\mathtt{t} = !l \qquad \mathtt{t}' = \mu(l) \qquad \mu' = \mu$

By part (f) of the inversion lemma, $\mathtt{T} = \mathtt{T}_{11}$, and $\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ \mathtt{T}_{11}$. By part (h) of the inversion lemma, $\mathtt{T}_{11} = \mathtt{Ref}\ \Sigma(l)$, i.e., $\mathtt{T} = \mathtt{T}_{11} = \Sigma(l)$. But now, from the assumption that $\Gamma \mid \Sigma \vdash \mu$, we can conclude (by the definition of $\Gamma \mid \Sigma \vdash \mu$) that $\Gamma \mid \Sigma \vdash \mu(l) : \Sigma(l)$.

the skeleton is written:

Case E-DEREFLOC*:* $\mathtt{t} = !l \qquad \mathtt{t}' = \mu(l) \qquad \mu' = \mu$

Inversion(f), inversion(h)

As a second example, the case for E-REF is also given below.

THEOREM [PRESERVATION]: If

$\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$
$\Gamma \mid \Sigma \vdash \mu$ (i.e., $dom(\mu) = dom(\Sigma)$ and $\Gamma \mid \Sigma \vdash \mu(l) : \Sigma(l)$ for every $l \in dom(\mu)$)
$\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$

then, for some $\Sigma' \supseteq \Sigma$,

$\Gamma \mid \Sigma' \vdash \mathtt{t}' : \mathtt{T}$
$\Gamma \mid \Sigma' \vdash \mu'$.

Proof: By induction on evaluation derivations, with a case analysis on the final rule used.

Case E-APP1*:* $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \quad \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu' \quad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2$

Answer: Inversion(c), IH, weakening, T-APP.

Case E-APP2*:*

Similar.

Case E-APPABS*:* $\mathtt{t} = (\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \quad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \mu' = \mu$

Answer: inversion(c), inversion(b), substitution

Case E-REF*:* $\mathtt{t} = \mathtt{ref}\ \mathtt{t}_1 \quad \mathtt{t}' = \mathtt{ref}\ \mathtt{t}_1' \quad \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'$

inversion(e), IH, T-REF

Case E-DEREFLOC*:* $\mathtt{t} = !l \quad \mathtt{t}' = \mu(l) \quad \mu' = \mu$

Inversion(f), inversion(h)

Case E-DEREF*:* $!\mathtt{t}_1 \mid \mu \longrightarrow\ !\mathtt{t}_1' \mid \mu'$

Answer: inversion(f), IH, T-DEREF

Case E-ASSIGN*:* $\mathtt{t} = l{:=}\mathtt{v}_2 \quad \mathtt{t}' = \mathtt{unit} \quad \mu' = [l \mapsto \mathtt{v}_2]\mu$

Answer: inversion(g), inversion(h), replacement, T-UNIT

Case E-ASSIGN1*:* $\mathtt{t} = \mathtt{t}_1{:=}\mathtt{t}_2 \quad \mathtt{t}' = \mathtt{t}_1'{:=}\mathtt{t}_2 \quad \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'$

Answer: inversion(g), IH, weakening, T-ASSIGN

Case E-ASSIGN2*:*

Similar.

Grading scheme: 4 points possible for each part

- *1 point off for missing weakening, inversion, or use of a typing rule*
- *2 points off for missing IH, substitution, or replacement*
- *1 point off for wrong ordering*
- *1 point off for using the wrong case of the inversion lemma*
- *1 point off for including an unnecessary step, as long the extra step was possible; 2 points if the extra step was actually wrong (not legal at the point where it appeared)*
- *3 points off whole problem for writing a full proof instead of a sketch*

Subtyping

The definition of the simply typed lambda-calculus with records and subtyping is reproduced for your reference on page 13.

6. (11 points) For each type S from the left-hand column below, draw a line connecting it to each type T in the right-hand column such that S <: T.

Choices for S:	Choices for T:
`{a:{}, b:{x:Top}}`	`({}→{a:Top})→{}`
`Top→Top`	`Top→Top`
`{}→{}`	`{}→Top`
`Top`	`Top→{}`
`({a:Top}→{})→{b:Top}`	`{b:Top}`
`{b:Top→Top}`	`{b:{}}`

Answer: Numbering both columns from top to bottom, we have

- S_1 <: T_5, T_6
- S_2 <: T_2, T_3
- S_3 <: T_3
- S_4 *is not a subtype of any of the* T*s*
- S_5 <: T_1
- S_6 <: T_5

Grading scheme: 1 point off for each missing line; 1 off for each incorrect line.

7. (12 points) It is easy to show, by induction on subtyping derivations, that

   LEMMA A: If $\mathsf{Top} <: \mathsf{T}$, then $\mathsf{T} = \mathsf{Top}$.

   A similar, but slightly more interesting, lemma holds for supertypes of arrow types.

   LEMMA B: If $\mathsf{S}_1 \to \mathsf{S}_2 <: \mathsf{T}$, then either $\mathsf{T} = \mathsf{Top}$ or else T has the form $\mathsf{T}_1 \to \mathsf{T}_2$, with $\mathsf{T}_1 <: \mathsf{S}_1$ and $\mathsf{S}_2 <: \mathsf{T}_2$.

   Fill in the arguments for the S-ARROW and S-TRANS cases of its proof.

   Proof: By induction on subtyping derivations. Proceed by a case analysis on the last rule used in the derivation.

   Case S-REFL*:* $\mathsf{T} = \mathsf{S}_1 \to \mathsf{S}_2$

   T clearly has the required form, with $\mathsf{T}_1 = \mathsf{S}_1$ and $\mathsf{T}_2 = \mathsf{S}_2$. The inclusions $\mathsf{T}_1 <: \mathsf{S}_1$ and $\mathsf{S}_2 <: \mathsf{T}_2$ both follow by S-REFL.

   Case S-TRANS*:* $\mathsf{S}_1 \to \mathsf{S}_2 <: \mathsf{U}$ $\quad \mathsf{U} <: \mathsf{T}$

   Answer:
   By the induction hypothesis, either $U = Top$ or else U has the form $U_1 \to U_2$, with $U_1 <: S_1$ and $S_2 <: U_2$. In the first case ($U = Top$), lemma A *tells us that $T = Top$ and we are finished. Otherwise, $U = U_1 \to U_2$, and we can use the induction hypothesis again to show that either $T = Top$ or else T has the form $T_1 \to T_2$, with $T_1 <: U_1$ and $U_2 <: T_2$. But now, from $T_1 <: U_1$ and $U_1 <: S_1$, we obtain $T_1 <: S_1$ using* S-TRANS*. Similarly, from $S_2 <: U_2$ and $U_2 <: T_2$,* S-TRANS *yields $S_2 <: T_2$.*

   Case S-ARROW*:* $\mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2$ $\quad \mathsf{T}_1 <: \mathsf{S}_1$ and $\mathsf{S}_2 <: \mathsf{T}_2$

   Answer:
   Immediate.

   Case S-TOP*:* $\mathsf{T} = \mathsf{Top}$

   Immediate.

   Case S-RCDWIDTH, S-RCDDEPTH, S-RCDPERM, S-TOP*:*

   Can't happen: T has the wrong form.

   Grading scheme: The subcase S-TRANS *was worth 9 points; subcase* S-ARROW*, 3 points.*

   Minus 1 point for each case when, in an otherwise correct proof, a step was not justified by an explicit reference to an inductive case, Lemma A, or the rule S-TRANS*. Minus 1 point when, instead of IH, the proof referred to Lemma B.*

   Minus 3 points when the consideration of the Top subcase in the induction step is omitted completely.

   At most 4 points (out of 9) were given when the induction argument was very unclear but most of the statements that would arise in a correct proof were mentioned.

   Some solutions got 0 points for totally incoherent arguments.

8. (9 points) Suppose we remove rule S-Arrow from the subtype relation. Which of the following properties will remain true? For each one, write either "true" (if it remains true) or else "false" (if it becomes false), *plus* (in either case) a one-sentence justification of your answer.

   (a) Existence of minimal types (if term `t` is typable in context Γ, then there is some type S such that $\Gamma \vdash$ `t : S` and, for every type T such that $\Gamma \vdash$ `t : T`, we have `S <: T`)

   Answer: False. For example, the term λ`x:{}. x` has both the types {}→ {} and {}→ Top, but, without the arrow rule, these two types have no common lower bound.

   (b) Progress (if `t` is a closed, well-typed term, then either `t` is a value or else $t \longrightarrow t'$ for some t')

   Answer: True: removing pairs from the subtype relation can only reduce *the number of well-typed terms, which can only make it* easier *for progress to hold.*

   (c) Preservation (if `t` has type T and $t \longrightarrow t'$, then t' also has type T)

   Answer: True. The only part of the preservation proof that changes is the inversion lemma (which changes back to its form from chapter 9(!) and becomes easier to prove) and the case of the main proof where it is used.

   This part of the question was more subtle than the others, since it is not *the case that preservation remains true for any simple reason. In particular, it's not correct to observe, at this point, that making fewer terms well typed can only make preservation easier by weakening its premise, since its conclusion is also weakened. Indeed, it could very well be that removing things from the subtype relation could break preservation; it just happens that, in this case, it does not.*

Grading scheme: -3 for each wrong answer. One point awarded for correct answer. One point awarded for partial explanation (given generously). One point awarded for complete explanation (given sparingly).

For reference: Simply typed lambda calculus with `Unit`

Syntax

t ::=		*terms*
	unit	*constant* unit
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
v ::=		*values*
	unit	*constant unit*
	λx:T.t	*abstraction value*
T ::=		*types*
	Unit	*unit type*
	T→T	*type of functions*
Γ ::=		*contexts*
	∅	*empty context*
	Γ, x:T	*term variable binding*

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \qquad \text{(E-AppAbs)}$$

Typing $\boxed{\Gamma \vdash t : T}$

$$\Gamma \vdash \mathtt{unit} : \mathtt{Unit} \qquad \text{(T-Unit)}$$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \qquad \text{(T-App)}$$

For reference: References

New syntactic forms

$\mathtt{t}$	::=	...	*terms*
		$\mathtt{ref\ t}$	*reference creation*
		$\mathtt{!t}$	*dereference*
		$\mathtt{t{:}{=}t}$	*assignment*
		l	*store location*
$\mathtt{v}$	::=	...	*values*
		l	*store location*
$\mathtt{T}$	::=	...	*types*
		$\mathtt{Ref\ T}$	*type of reference cells*
μ	::=	...	*stores*
		$\emptyset$	*empty store*
		$\mu, l = \mathtt{v}$	*location binding*
Σ	::=	...	*store typings*
		$\emptyset$	*empty store typing*
		$\Sigma, l{:}\mathtt{T}$	*location typing*

New evaluation rules $\boxed{\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'}$

$$\frac{\mathtt{t}_1 | \mu \longrightarrow \mathtt{t}_1' | \mu'}{\mathtt{t}_1\ \mathtt{t}_2 | \mu \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2 | \mu'} \qquad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 | \mu \longrightarrow \mathtt{t}_2' | \mu'}{\mathtt{v}_1\ \mathtt{t}_2 | \mu \longrightarrow \mathtt{v}_1\ \mathtt{t}_2' | \mu'} \qquad \text{(E-APP2)}$$

$$(\lambda \mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 | \mu \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} | \mu \qquad \text{(E-APPABS)}$$

$$\frac{l \notin dom(\mu)}{\mathtt{ref}\ \mathtt{v}_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto \mathtt{v}_1)} \qquad \text{(E-REFV)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{ref}\ \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{ref}\ \mathtt{t}_1' \mid \mu'} \qquad \text{(E-REF)}$$

$$\frac{\mu(l) = \mathtt{v}}{\mathtt{!}l \mid \mu \longrightarrow \mathtt{v} \mid \mu} \qquad \text{(E-DEREFLOC)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{!t}_1 \mid \mu \longrightarrow \mathtt{!t}_1' \mid \mu'} \qquad \text{(E-DEREF)}$$

$$l{:}{=}\mathtt{v}_2 \mid \mu \longrightarrow \mathtt{unit} \mid [l \mapsto \mathtt{v}_2]\mu \qquad \text{(E-ASSIGN)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1{:}{=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'{:}{=}\mathtt{t}_2 \mid \mu'} \qquad \text{(E-ASSIGN1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1\mathtt{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1\mathtt{:=}\mathtt{t}_2' \mid \mu'} \qquad \text{(E-ASSIGN2)}$$

New typing rules $\boxed{\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}}$

$$\Gamma \mid \Sigma \vdash \mathtt{unit} : \mathtt{Unit} \qquad \text{(T-UNIT)}$$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \mid \Sigma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \mid \Sigma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

$$\frac{\Sigma(l) = \mathtt{T}_1}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ \mathtt{T}_1} \qquad \text{(T-LOC)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_1} \qquad \text{(T-REF)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{!}\mathtt{t}_1 : \mathtt{T}_{11}} \qquad \text{(T-DEREF)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1\mathtt{:=}\mathtt{t}_2 : \mathtt{Unit}} \qquad \text{(T-ASSIGN)}$$

For reference: Simply typed lambda calculus with records and subtyping

New syntactic forms

$\mathtt{t}$	::=	...	*terms*
		$\{\mathtt{l}_i\mathtt{=t}_i{}^{\,i\in 1..n}\}$	*record*
		$\mathtt{t.l}$	*projection*
$\mathtt{v}$	::=	...	*values*
		$\{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..n}\}$	*record value*
$\mathtt{T}$	::=	...	*types*
		$\{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}$	*type of records*
		$\mathtt{Top}$	*maximum type*

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..n}\}.\mathtt{l}_j \longrightarrow \mathtt{v}_j \qquad \text{(E-ProjRcd)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.\mathtt{l} \longrightarrow \mathtt{t}_1'.\mathtt{l}} \qquad \text{(E-Proj)}$$

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}_j'}{\begin{array}{c}\{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..j-1},\mathtt{l}_j\mathtt{=t}_j,\mathtt{l}_k\mathtt{=t}_k{}^{\,k\in j+1..n}\}\\ \longrightarrow \{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..j-1},\mathtt{l}_j\mathtt{=t}_j',\mathtt{l}_k\mathtt{=t}_k{}^{\,k\in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

New subtyping rules $\boxed{\mathtt{S} <: \mathtt{T}}$

$$\mathtt{S} <: \mathtt{S} \qquad \text{(S-Refl)}$$

$$\frac{\mathtt{S} <: \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}} \qquad \text{(S-Trans)}$$

$$\mathtt{S} <: \mathtt{Top} \qquad \text{(S-Top)}$$

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \qquad \mathtt{S}_2 <: \mathtt{T}_2}{\mathtt{S}_1 \to \mathtt{S}_2 <: \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(S-Arrow)}$$

$$\{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n+k}\} <: \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\} \qquad \text{(S-RcdWidth)}$$

$$\frac{\text{for each } i \quad \mathtt{S}_i <: \mathtt{T}_i}{\{\mathtt{l}_i\mathtt{:S}_i{}^{\,i\in 1..n}\} <: \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

$$\frac{\{\mathtt{k}_j\mathtt{:S}_j{}^{\,j\in 1..n}\} \text{ is a permutation of } \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}}{\{\mathtt{k}_j\mathtt{:S}_j{}^{\,j\in 1..n}\} <: \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}} \qquad \text{(S-RcdPerm)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\text{for each } i \quad \Gamma \vdash \mathtt{t}_i : \mathtt{T}_i}{\Gamma \vdash \{\mathtt{l}_i\mathtt{=t}_i{}^{\,i\in 1..n}\} : \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\}}{\Gamma \vdash \mathtt{t}_1.\mathtt{l}_j : \mathtt{T}_j} \qquad \text{(T-Proj)}$$

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{S} \qquad \mathtt{S} <: \mathtt{T}}{\Gamma \vdash \mathtt{t} : \mathtt{T}} \qquad \text{(T-Sub)}$$