CIS 500 — Software Foundations

Midterm II

November 13, 2002

Name: ______________________________

Student ID: ______________________________
(from your PennCard)

Email ______________________________

	Score
1	
2	
3	
4	
5	
6	
7	
8	
Total	

Instructions

- This is a closed-book exam: you may not make use of any books or notes.
- You have 80 minutes to answer all of the questions. The entire exam is worth 80 points.
- Questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.
- Partial credit will be given. All correct answers are short. The back side of each page may be used as a scratch pad.
- Good luck!

Simply typed lambda-calculus

The definition of the simply typed lambda-calculus with `Unit` is reproduced on page 15.

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)`

   (b) `λx. x unit x`

   (c) `λx. x unit unit`

   (d) `λx. λy. λz. (x y) (y z)`

References

The definition of the simply typed lambda-calculus with references is reproduced on page 15.

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
```
Result: Store:

(b)
```
let x = ref 2 in
let y = ref x in
let z = ref y in
!z
```
Result: Store:

(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
```

Result: Store:

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.

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;
```

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 $\Gamma \mid \Sigma \vdash \mathtt{x} : \mathsf{T}$, then $\mathtt{x}{:}\mathsf{T} \in \Gamma$.
(b) If $\Gamma \mid \Sigma \vdash \lambda\mathtt{x}{:}\mathsf{T}_1.\ \mathtt{t}_2 : \mathsf{T}$, then $\mathsf{T} = \mathsf{T}_1 \to \mathsf{T}_2$ for some T_2 with $\Gamma, \mathtt{x}{:}\mathsf{T}_1 \mid \Sigma \vdash \mathtt{t}_2 : \mathsf{T}_2$.
(c) If $\Gamma \mid \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathsf{T}$, then there is some type T_{11} such that $\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathsf{T}_{11} \to \mathsf{T}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathsf{T}_{11}$.
(d) If $\Gamma \mid \Sigma \vdash \mathtt{unit} : \mathsf{T}$, then $\mathsf{T} = \mathsf{Unit}$.
(e) If $\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathsf{T}$, then $\mathsf{T} = \mathsf{Ref}\ \mathsf{T}_1$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathsf{T}_1$.
(f) If $\Gamma \mid \Sigma \vdash\ !\mathtt{t}_1 : \mathsf{T}$, then $\mathsf{T} = \mathsf{T}_{11}$ with $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathsf{Ref}\ \mathsf{T}_{11}$.
(g) If $\Gamma \mid \Sigma \vdash \mathtt{t}_1{:=}\mathtt{t}_2 : \mathsf{T}$, then $\mathsf{T} = \mathsf{Unit}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_1 \in \mathsf{Ref}\ \mathsf{T}_{11}$ and $\Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathsf{T}_{11}$.
(h) If $\Gamma \mid \Sigma \vdash l : \mathsf{T}$, then $\mathsf{T} = \mathsf{Ref}\ \Sigma(l)$.

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

LEMMA [REPLACEMENT]: If

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

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

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

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: $\quad \mathtt{t} = !l \qquad \mathtt{t}' = \mu(l) \qquad \mu' = \mu$

By part (f) of the inversion lemma, $\mathsf{T} = \mathsf{T}_{11}$, and $\Gamma \mid \Sigma \vdash l : \mathsf{Ref}\ \mathsf{T}_{11}$. By part (h) of the inversion lemma, $\mathsf{T}_{11} = \mathsf{Ref}\ \Sigma(l)$, i.e., $\mathsf{T} = \mathsf{T}_{11} = \Sigma(l)$. 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: $\quad \mathtt{t} = !l \qquad \mathtt{t}' = \mu(l) \qquad \mu' = \mu$

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*:* $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2$ $\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'$ $\mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2$

Case E-APP2*:*
Similar.

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

Case E-REF*:* $\mathtt{t} = \mathtt{ref}\ \mathtt{t}_1$ $\mathtt{t}' = \mathtt{ref}\ \mathtt{t}_1'$ $\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'$
inversion(e), IH, T-REF

Case E-DEREFLOC*:* $\mathtt{t} = !l$ $\mathtt{t}' = \mu(l)$ $\mu' = \mu$
Inversion(f), inversion(h)

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

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

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

Case E-ASSIGN2*:*
Similar.

Subtyping

The definition of the simply typed lambda-calculus with records and subtyping is reproduced for your reference on page 17.

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:{}}`

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 $\mathsf{S_1 \to S_2} <: \mathsf{T}$, then either $\mathsf{T} = \mathsf{Top}$ or else T has the form $\mathsf{T_1 \to T_2}$, 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*:* $\mathsf{T} = \mathsf{S_1 \to S_2}$

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*:* $\mathsf{S_1 \to S_2} <: \mathsf{U}$ $\quad \mathsf{U} <: \mathsf{T}$

Fill in:

Case S-ARROW*:* $\mathsf{T} = \mathsf{T_1 \to T_2}$ $\quad \mathsf{T_1} <: \mathsf{S_1}$ and $\mathsf{S_2} <: \mathsf{T_2}$

Fill in:

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.

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 $\mathtt{t}$ is typable in context Γ, then there is some type S such that $\Gamma \vdash \mathtt{t} : \mathsf{S}$ and, for every type T such that $\Gamma \vdash \mathtt{t} : \mathsf{T}$, we have $\mathsf{S} <: \mathsf{T}$)

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

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

For reference: Simply typed lambda calculus with `Unit`

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{unit}$	*constant* `unit`
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction*
	$\mathtt{t}\ \mathtt{t}$	*application*
$\mathtt{v}$ ::=		*values*
	$\mathtt{unit}$	*constant unit*
	$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction value*
$\mathtt{T}$ ::=		*types*
	$\mathtt{Unit}$	*unit type*
	$\mathtt{T}{\to}\mathtt{T}$	*type of functions*
Γ ::=		*contexts*
	$\varnothing$	*empty context*
	$\Gamma, \mathtt{x}{:}\mathtt{T}$	*term variable binding*

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

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \qquad \text{(E-App2)}$$

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

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

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

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

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

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

For reference: References

New syntactic forms

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

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

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2 \mid \mu'} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1\ \mathtt{t}_2' \mid \mu'} \quad \text{(E-App2)}$$

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

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

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

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

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

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

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'{:=}\mathtt{t}_2 \mid \mu'} \quad \text{(E-Assign1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1 \mathtt{:=} \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1 \mathtt{:=} \mathtt{t}_2' \mid \mu'} \tag{E-Assign2}$$

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

$$\Gamma \mid \Sigma \vdash \mathtt{unit} : \mathtt{Unit} \tag{T-Unit}$$

$$\frac{\mathtt{x}:\mathtt{T} \in \Gamma}{\Gamma \mid \Sigma \vdash \mathtt{x} : \mathtt{T}} \tag{T-Var}$$

$$\frac{\Gamma, \mathtt{x}:\mathtt{T}_1 \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \mid \Sigma \vdash \lambda \mathtt{x}:\mathtt{T}_1 . \mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \tag{T-Abs}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \tag{T-App}$$

$$\frac{\Sigma(l) = \mathtt{T}_1}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ \mathtt{T}_1} \tag{T-Loc}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_1} \tag{T-Ref}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{!}\mathtt{t}_1 : \mathtt{T}_{11}} \tag{T-Deref}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1 \mathtt{:=} \mathtt{t}_2 : \mathtt{Unit}} \tag{T-Assign}$$

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

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

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

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