CIS 500 — Software Foundations

Midterm II

November 16, 2005

Name: ______________________________

Email ______________________________

Status ____ registered for the course

____ not registered

	Score
1	/ 10
2	/ 10
3	/ 10
4	/ 5
5	/ 5
6	/ 10
7	/ 20
8	/ 10
Total	/80

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 values of questions are not always proportional to their difficulty. Do not spend too much time on any one question.
- Partial credit will be given wherever possible. 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 following questions refer to the simply typed lambda-calculus with booleans and error handling. The syntax, typing, and evaluation rules for this system are given on page 1 of the companion handout.

1. (10 points) Write down the types of each of the following terms. If a term can be given many types, you should write down the *smallest* one. If a term does not type check, write NONE. Note: Recall that $\mathtt{T} \to \mathtt{T} \to \mathtt{T}$ is parsed as $\mathtt{T} \to (\mathtt{T} \to \mathtt{T})$.

   (a) λx:Bool→Bool. x (x (x (x (x true))))

   Type: ______________________________

   (b) (λx:Bool. λy:Bool→Bool→Bool. true) false (λz:Bool→Bool. true)

   Type: ______________________________

   (c) (λx:Bool. λy:Bool. error) false false false false false

   Type: ______________________________

   (d) λx:Bool→Bool→Bool. x (x error)

   Type: ______________________________

   (e) try (if (λx:Bool. x) error then (error false) else error) with λy:Bool→Bool. y

   Type: ______________________________

References

The following questions refer to the simply typed lambda-calculus with references. The syntax, typing, and evaluation rules for this system are given on page 3 of the companion handout.

2. (10 points) Which of the following functions *could* evaluate to 42 when applied to a *single* argument and evaluated with a store of the appropriate type? Circle YES and give the argument and store if that is the case, and circle NO otherwise.

   For example, the term

   $$\lambda \texttt{x:Ref Nat. !x + 1}$$

   evaluates to 42 with argument $\texttt{l}_1$ and store $(\texttt{l}_1 \mapsto \texttt{41})$.

   (a) $\lambda \texttt{x:Ref Nat. x}$

   YES, with argument ______________________ and store ______________________________
   NO

   (b) $\lambda \texttt{x:Ref Nat. (x:= 3; } \texttt{l}_1 \texttt{ := 42; !x)}$

   YES, with argument ______________________ and store ______________________________
   NO

   (c) $\lambda \texttt{f:Unit} \rightarrow \texttt{Unit. (} \texttt{l}_1 \texttt{ := 3; f unit; !} \texttt{l}_1 \texttt{)}$

   YES, with argument ______________________ and store ______________________________
   NO

3. (10 points) Suppose we add an increment operator (`t++`) to the simply typed lambda-calculus with references. This operator should increase the value stored in a numerical reference by one. For example, the result of evaluating the following term

```
let x = ref 3 in ( x++ ; !x )
```

with the empty store is the value 4.

We start formalizing this idea by adding a new term form for the increment operator:

```
t ++
```

and a new computation rule for this new term form.

$$\frac{\mu(l) = \texttt{nv}}{\texttt{l++} \mid \mu \longrightarrow \texttt{unit} \mid [\texttt{l} \mapsto \texttt{succ nv}]\mu} \text{ L-IncrLoc}$$

(a) What congruence rule(s) should we add?

(b) What typing rule(s) should we add?

Implementing a type checker

Consider an implementation of a type checker for the simply-typed lambda-calculus extended with sums. The syntax of types will be extended in the following way:

```
type ty =
  ...
  TySum of ty * ty
```

The syntax of terms will be extended in the following way:

```
type term =
  ...
  TmCase of info * term * (string * term) * (string * term)
  TmInl of info * term * ty
  TmInr of info * term * ty
```

Your job is to finish the implementation of the function

```
typeof : context → term → ty
```

This recursive function returns the type of a term in a particular context. The general form of the function is a pattern match on the the form of the term.

```
let rec typeof (ctx:context) (t:term) :ty =
  match t with
    ...          (* Branches for variables, abstractions, applications *)
   | TmInl(info, t0, tyAnnot) → ... (* Branch for inl *)
   | TmInr(info, t0, tyAnnot) →  ...(* Branch for inr *)
   | TmCase(info, t0, (x1, t1), (x2, t2)) →  ... (* Branch for case *)
```

In this implementation, a `context` is composed of variable bindings and has the following interface:

```
type binding = VarBind of ty
val emptycontext : context
val addbinding   : context → string → binding → context
val getbinding   : info → context → int → binding
```

4. (5 points) Recall the typing rule for `inl` expressions:

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \vdash \mathtt{inl}\ \mathtt{t}_1\ \mathtt{as}\ \mathtt{T}_1\mathtt{+}\mathtt{T}_2 : \mathtt{T}_1\mathtt{+}\mathtt{T}_2} \qquad \text{(T-INL)}$$

One of the following segments of OCaml code correctly implements this rule. Circle the letter associated with the correct answer. The important differences are underlined.

(a)
```
TmInl(fi, t1, tyAnnot) →
    (match typeof ctx t1 with
      TySum(ty1, ty2) →
        if tyAnnot = ty1 then ty1
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(b)
```
TmInl(fi, t1, tyAnnot) →
    (match tyAnnot with
      TySum(ty1, ty2) →
        if typeof ctx t1 = ty1 then tyAnnot
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(c)
```
TmInl(fi, t1, tyAnnot) →
    (match typeof ctx t1 with
      TySum(ty1, ty2) →
        if tyAnnot = ty1 then tyAnnot
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(d)
```
TmInl(fi, t1, tyAnnot) →
    (match tyAnnot with
      TySum(ty1, ty2) →
        if ty1 = ty2 then typeof ctx t1
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

5. (5 points) Recall the typing rule for case expressions:

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{T}_1\mathtt{+}\mathtt{T}_2 \qquad \Gamma, \mathtt{x}_1 \mathtt{:} \mathtt{T}_1 \vdash \mathtt{t}_1 : \mathtt{T} \qquad \Gamma, \mathtt{x}_2 \mathtt{:} \mathtt{T}_2 \vdash \mathtt{t}_2 : \mathtt{T}}{\Gamma \vdash \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{inl}\ \mathtt{x}_1 \Rightarrow \mathtt{t}_1 \mid \mathtt{inr}\ \mathtt{x}_2 \Rightarrow \mathtt{t}_2 : \mathtt{T}} \qquad \text{(T-CASE)}$$

One of the following segments of OCaml code correctly implements this rule. Circle the letter associated with the correct answer. The important differences are underlined.

(a)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
    (match (typeof ctx t0) with
      TySum(ty1, ty2) →
        let tyLcase = typeof ctx t1 in
        let tyRcase = typeof ctx t2 in
        if tyLcase = tyRcase then tyLcase
        else error fi "Branches of case have different types"
    | _ → error fi "Expected sum type")
```

(b)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
    (match (typeof ctx t0) with
      TySum(ty1, ty2) →
        let ctx' = addbinding ctx x1 (VarBind(typeof ctx t1)) in
        let ctx'' = addbinding ctx' x2 (VarBind(typeof ctx t2)) in
        let tyLcase = typeof ctx'' t1 in
        let tyRcase = typeof ctx'' t2 in
        if tyLcase = tyRcase then tyLcase
        else error fi "Branches of case have different types"
    | _ → error fi "Expected sum type")
```

(c)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
    (match (typeof ctx t0) with
      TySum(ty1, ty2) →
        let ctx' = addbinding ctx x1 (VarBind(ty1)) in
        let ctx'' = addbinding ctx' x2 (VarBind(ty2)) in
        let tyLcase = typeof ctx'' t1 in
        let tyRcase = typeof ctx'' t2 in
        if tyLcase = tyRcase then tyLcase
        else error fi "Branches of case have different types"
    | _ → error fi "Expected sum type")
```

(d)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
    (match (typeof ctx t0) with
      TySum(ty1, ty2) →
        let ctx' = addbinding ctx x1 (VarBind(ty1)) in
        let ctx'' = addbinding ctx x2 (VarBind(ty2)) in
        let tyLcase = typeof ctx' t1 in
        let tyRcase = typeof ctx'' t2 in
        if tyLcase = tyRcase then tyLcase
        else error fi "Branches of case have different types"
    | _ → error fi "Expected sum type")
```

Proving type soundness

The following questions refer to the simply-typed λ-calculus with unit and fix. The syntax, typing, and evaluation rules for this system are given on page 6 of the companion handout.

6. (10 points) **Theorem (Preservation): If** $\Gamma \vdash \mathtt{t} : \mathtt{T}$ **and** $\mathtt{t} \longrightarrow \mathtt{t}'$ **then** $\Gamma \vdash \mathtt{t}' : \mathtt{T}$**.**

   Consider a proof of this theorem by induction on the typing derivation, $\Gamma \vdash \mathtt{t} : \mathtt{T}$. Show the case that occurs when the derivation ends with the rule T-FIX. In your proof you may use any of the following lemmas: *canonical forms*, *substitution*, *weakening*, *permutation*, and *inversion of typing*. These lemmas are listed in the companion handout on page 7. Be explicit about each step of the proof, and do not include any irrelevant information.

Properties of Typed Languages

The following questions refer to the simply typed λ-calculus with products and `Bool`. The syntax, typing, and evaluation rules for this system are given on page 8 of the companion handout.

7. (20 points) Recall the following theorems about the simply typed λ-calculus with products and `Bool`:

   - **Progress**: If $\vdash \texttt{t} : \texttt{T}$, then either `t` is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$.
   - **Preservation**: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{T}$.
   - **Uniqueness of types**: Each term `t` has at most one type, and if `t` has a type, then there is exactly one derivation of that typing.

   In each problem below ((a) through (c)), we add a single rule to this language. Consider these additions separately. For each theorem, circle whether the theorem remains true or if it becomes false. If a theorem becomes false, give a counterexample showing why.

   (a) $$\frac{}{\Gamma \vdash \{\texttt{t}_1\texttt{,}\ \texttt{t}_2\} : \texttt{Nat}}\ \text{T-PairNat}$$

   Progress: TRUE FALSE, because...

   Preservation: TRUE FALSE, because...

   Uniqueness of types: TRUE FALSE, because...

   (b) $$\frac{}{\texttt{pred}\ \{\texttt{t}_1\texttt{,}\ \texttt{t}_2\} \longrightarrow \texttt{t}_1}\ \text{E-PredPair}$$

   Progress: TRUE FALSE, because...

   Preservation: TRUE FALSE, because...

   Uniqueness of types: TRUE FALSE, because...

(c) $$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{fst}\,\{\mathtt{t}_1, \mathtt{t}_2\} : \mathtt{Nat}} \text{ T-FSTNAT}$$

Progress: TRUE FALSE, because...

Preservation: TRUE FALSE, because...

Uniqueness of types: TRUE FALSE, because...

Derived forms

The following questions refer to the simply typed λ-calculus with products and `Bool`*. The syntax, typing, and evaluation rules for this system are given on page 8 of the companion handout.*

8. (10 points) Let λ^I be the simply typed λ-calculus with products and `Bool`. We extend λ^I to a language that we call λ^E which has a new "`and`" construct as follows.

New syntax:

$$\begin{array}{lcll} \texttt{t} & ::= & \ldots & \text{terms:} \\ & & \texttt{and t t} & \text{conjunction} \end{array}$$

New typing rules:

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Bool} \quad \Gamma \vdash \texttt{t}_2 : \texttt{Bool}}{\Gamma \vdash \texttt{and } \texttt{t}_1\ \texttt{t}_2 : \texttt{Bool}} \text{ T-AND}$$

New evaluation rules:

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{and } \texttt{t}_1\ \texttt{t}_2 \longrightarrow \texttt{and } \texttt{t}_1'\ \texttt{t}_2} \text{ E-AND1} \qquad \frac{\texttt{t}_2 \longrightarrow \texttt{t}_2'}{\texttt{and } \texttt{v}_1\ \texttt{t}_2 \longrightarrow \texttt{and } \texttt{v}_1\ \texttt{t}_2'} \text{ E-AND2}$$

$$\frac{}{\texttt{and true true} \longrightarrow \texttt{true}} \text{ E-ANDTRUE}$$

$$\frac{}{\texttt{and false } \texttt{v}_2 \longrightarrow \texttt{false}} \text{ E-ANDFALSE1} \qquad \frac{}{\texttt{and } \texttt{v}_1 \texttt{ false} \longrightarrow \texttt{false}} \text{ E-ANDFALSE2}$$

Rather than treat `and` as a primitive, we can try to make it a derived form by giving the following function e from λ^E to λ^I:

$$\begin{array}{rcl} e(\texttt{x}) & = & \texttt{x} \\ e(\lambda\texttt{x:T. t}) & = & \lambda\texttt{x:T. } e(\texttt{t}) \\ e(\texttt{t}_1\ \texttt{t}_2) & = & e(\texttt{t}_1)\ e(\texttt{t}_2) \\ e(\texttt{true}) & = & \texttt{true} \\ e(\texttt{false}) & = & \texttt{false} \\ e(\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3) & = & \texttt{if } e(\texttt{t}_1) \texttt{ then } e(\texttt{t}_2) \texttt{ else } e(\texttt{t}_3) \\ e(\texttt{t.1}) & = & (e(\texttt{t}))\texttt{.1} \\ e(\texttt{t.2}) & = & (e(\texttt{t}))\texttt{.2} \\ e(\{\texttt{t}_1\texttt{,}\ \texttt{t}_2\}) & = & \{e(\texttt{t}_1), e(\texttt{t}_2)\} \\ e(\texttt{and } \texttt{t}_1\ \texttt{t}_2) & = & \texttt{if } e(\texttt{t}_1) \texttt{ then } e(\texttt{t}_2) \texttt{ else false} \end{array}$$

Are we successful? For each of the following properties, circle TRUE if it holds for this "derived form" and otherwise circle FALSE and give a counterexample.

(a) if $\texttt{t} \rightarrow_E \texttt{t}'$ then $e(\texttt{t}) \rightarrow_I e(\texttt{t}')$.

TRUE FALSE, because...

(b) if $\Gamma \vdash^E \texttt{t} : \texttt{T}$ then $\Gamma \vdash^I e(\texttt{t}) : \texttt{T}$.

TRUE FALSE, because...

(c) if $\texttt{t}$ is a value then $e(\texttt{t})$ is a value.

TRUE FALSE, because...

Companion handout

Full definitions of the systems used in the exam

Simply-typed lambda calculus with error handling and `Bool`

Syntax

t ::=		*terms*
	error	*run-time error*
	true	*constant true*
	false	*constant false*
	if t then t else t	*conditional*
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	try t with t	*trap errors*
v ::=		*values*
	true	*true value*
	false	*false value*
	λx:T.t	*abstraction value*
T ::=		*types*
	T$\rightarrow$T	*type of functions*
	Bool	*type of booleans*
Γ ::=		*type environments*
	$\emptyset$	*empty type env.*
	Γ, x:T	*term variable binding*

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

$$\mathtt{if\ true\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_2} \quad \text{(E-IfTrue)}$$

$$\mathtt{if\ false\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_3} \quad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}} \quad \text{(E-If)}$$

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

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

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

$$\mathtt{try\ v_1\ with\ t_2} \longrightarrow \mathtt{v_1} \quad \text{(E-TryV)}$$

$$\mathtt{try\ error\ with\ t_2} \longrightarrow \mathtt{t_2} \quad \text{(E-TryError)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{try\ t_1\ with\ t_2} \longrightarrow \mathtt{try\ t_1'\ with\ t_2}} \quad \text{(E-Try)}$$

$$\mathtt{if\ error\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{error} \quad \text{(E-IfErr)}$$

$$\mathtt{error\ t_2} \longrightarrow \mathtt{error} \quad \text{(E-AppErr1)}$$

$$\mathtt{v_1\ error} \longrightarrow \mathtt{error} \quad \text{(E-AppErr2)}$$

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

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

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

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

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool} \tag{T-True}$$

$$\Gamma \vdash \mathtt{false} : \mathtt{Bool} \tag{T-False}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \tag{T-If}$$

$$\Gamma \vdash \mathtt{error} : \mathtt{T} \tag{T-Error}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}}{\Gamma \vdash \mathtt{try}\ \mathtt{t}_1\ \mathtt{with}\ \mathtt{t}_2 : \mathtt{T}} \tag{T-Try}$$

Simply-typed lambda calculus with references (and Unit, Nat, Bool)

Syntax

t ::=		*terms*
	unit	*constant* unit
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	ref t	*reference creation*
	!t	*dereference*
	t:=t	*assignment*
	l	*store location*
	true	*constant true*
	false	*constant false*
	if t then t else t	*conditional*
	0	*constant zero*
	succ t	*successor*
	pred t	*predecessor*
	iszero t	*zero test*
v ::=		*values*
	unit	*constant unit*
	λx:T.t	*abstraction value*
	l	*store location*
	true	*true value*
	false	*false value*
	nv	*numeric value*
T ::=		*types*
	Unit	*unit type*
	T$\to$T	*type of functions*
	Ref T	*type of reference cells*
	Bool	*type of booleans*
	Nat	*type of natural numbers*
μ ::=		*stores*
	$\emptyset$	*empty store*
	$\mu, l =$ v	*location binding*
Γ ::=		*type environments*
	$\emptyset$	*empty type env.*
	Γ, x:T	*term variable binding*
Σ ::=		*store typings*
	$\emptyset$	*empty store typing*
	Σ, l:T	*location typing*
nv ::=		*numeric values*

$\texttt{0}$	*zero value*
$\texttt{succ nv}$	*successor value*

Evaluation $\boxed{\texttt{t} \mid \mu \longrightarrow \texttt{t}' \mid \mu'}$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{t}_1\ \texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_1'\ \texttt{t}_2 \mid \mu'} \quad \text{(E-APP1)}$$

$$\frac{\texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_2' \mid \mu'}{\texttt{v}_1\ \texttt{t}_2 \mid \mu \longrightarrow \texttt{v}_1\ \texttt{t}_2' \mid \mu'} \quad \text{(E-APP2)}$$

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

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

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

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

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

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

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

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

$$\texttt{if true then t}_2\ \texttt{else t}_3 \mid \mu \longrightarrow \texttt{t}_2 \mid \mu \quad \text{(E-IFTRUE)}$$

$$\texttt{if false then t}_2\ \texttt{else t}_3 \mid \mu \longrightarrow \texttt{t}_3 \mid \mu \quad \text{(E-IFFALSE)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{if t}_1\ \texttt{then t}_2\ \texttt{else t}_3 \mid \mu \longrightarrow \texttt{if t}_1'\ \texttt{then t}_2\ \texttt{else t}_3 \mid \mu'} \quad \text{(E-IF)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{succ t}_1 \mid \mu \longrightarrow \texttt{succ t}_1' \mid \mu'} \quad \text{(E-SUCC)}$$

$$\texttt{pred 0} \mid \mu \longrightarrow \texttt{0} \mid \mu \quad \text{(E-PREDZERO)}$$

$$\texttt{pred (succ nv}_1\texttt{)} \mid \mu \longrightarrow \texttt{nv}_1 \mid \mu \quad \text{(E-PREDSUCC)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{pred t}_1 \mid \mu \longrightarrow \texttt{pred t}_1' \mid \mu'} \quad \text{(E-PRED)}$$

$$\texttt{iszero 0} \mid \mu \longrightarrow \texttt{true} \mid \mu \quad \text{(E-ISZEROZERO)}$$

$$\texttt{iszero (succ nv}_1\texttt{)} \mid \mu \longrightarrow \texttt{false} \mid \mu \quad \text{(E-ISZEROSUCC)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{iszero t}_1 \mid \mu \longrightarrow \texttt{iszero t}_1' \mid \mu'} \quad \text{(E-ISZERO)}$$

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

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

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

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

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

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

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

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

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

$$\Gamma \mid \Sigma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-TRUE)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \mid \Sigma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-IF)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{0} : \mathtt{Nat} \qquad \text{(T-ZERO)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{succ}\ \mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-SUCC)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{pred}\ \mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-PRED)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{iszero}\ \mathtt{t}_1 : \mathtt{Bool}} \qquad \text{(T-ISZERO)}$$

Simply-typed lambda calculus with Unit and fix

Syntax

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

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

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

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

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

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{fix t}_1 \longrightarrow \texttt{fix t}_1'} \qquad \text{(E-Fix)}$$

$$\texttt{fix}\ (\lambda\texttt{x:T}_1\texttt{.t}_2) \longrightarrow [\texttt{x} \mapsto \texttt{fix}\ (\lambda\texttt{x:T}_1\texttt{.t}_2)]\texttt{t}_2 \qquad \text{(E-FixBeta)}$$

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

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

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

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

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1\rightarrow\texttt{T}_1}{\Gamma \vdash \texttt{fix t}_1 : \texttt{T}_1} \qquad \text{(T-Fix)}$$

Properties of STLC + $unit$ + fix

1. *Lemma [Canonical forms]:*
   (a) If $\mathtt{v}$ is a value of type $\mathtt{T_1{\to}T_2}$, then $\mathtt{v}$ has the form $\lambda\mathtt{x{:}T_1.t_2}$.
   (b) If $\mathtt{v}$ is a value of type $\mathtt{Unit}$, then $\mathtt{v}$ is $\mathtt{unit}$.
2. *Lemma [Substitution]:* If $\Gamma, \mathtt{x{:}S} \vdash \mathtt{t : T}$ and $\Gamma \vdash \mathtt{s : S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t : T}$.
3. *Lemma [Permutation]:* If $\Gamma \vdash \mathtt{t : T}$ and Δ is a permutation of Γ then $\Delta \vdash \mathtt{t : T}$. Moreover the latter derivation has the same depth as the former.
4. *Lemma [Weakening]:* If $\Gamma \vdash \mathtt{t : T}$ and $\mathtt{x} \notin \mathit{dom}(\Gamma)$ then $\Gamma, \mathtt{x : S} \vdash \mathtt{t : T}$. Moreover the latter derivation has the same depth as the former.
5. *Lemma [Inversion of typing]:*
   (a) If $\Gamma \vdash \mathtt{x : R}$, then $\mathtt{x{:}R} \in \Gamma$.
   (b) If $\Gamma \vdash \lambda\mathtt{x{:}T_1.t_2 : R}$, then $\mathtt{R} = \mathtt{T_1{\to}R_2}$ for some $\mathtt{R_2}$ with $\Gamma, \mathtt{x{:}T_1} \vdash \mathtt{t_2 : R_2}$.
   (c) If $\Gamma \vdash \mathtt{t_1\ t_2 : R}$, then there is some type $\mathtt{T_{11}}$ such that $\Gamma \vdash \mathtt{t_1 : T_{11}{\to}R}$ and $\Gamma \vdash \mathtt{t_2 : T_{11}}$.
   (d) If $\Gamma \vdash \mathtt{unit : R}$, then $\mathtt{R} = \mathtt{Unit}$.
   (e) If $\Gamma \vdash \mathtt{fix\,t_1 : R}$, then $\Gamma \vdash \mathtt{t_1 : R \to R}$,

Simply-typed lambda calculus with products and Bool

Syntax

t ::=		*terms*
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	true	*constant true*
	false	*constant false*
	if t then t else t	*conditional*
	{t,t}	*pair*
	t.1	*first projection*
	t.2	*second projection*
v ::=		*values*
	λx:T.t	*abstraction value*
	true	*true value*
	false	*false value*
	{v,v}	*pair value*
T ::=		*types*
	T→T	*type of functions*
	Bool	*type of booleans*
	$T_1 \times T_2$	*product type*
Γ ::=		*type environments*
	∅	*empty type env.*
	Γ, x:T	*term variable binding*

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-APP2)}$$

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

$$\text{if true then } t_2 \text{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IFTRUE)}$$

$$\text{if false then } t_2 \text{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IFFALSE)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \longrightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \qquad \text{(E-IF)}$$

$$\{v_1,v_2\}.1 \longrightarrow v_1 \qquad \text{(E-PAIRBETA1)}$$

$$\{v_1,v_2\}.2 \longrightarrow v_2 \qquad \text{(E-PAIRBETA2)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \qquad \text{(E-PROJ1)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\mathtt{.2} \longrightarrow \mathtt{t}_1'\mathtt{.2}} \qquad \text{(E-Proj2)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\{\mathtt{t}_1,\mathtt{t}_2\} \longrightarrow \{\mathtt{t}_1',\mathtt{t}_2\}} \qquad \text{(E-Pair1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\{\mathtt{v}_1,\mathtt{t}_2\} \longrightarrow \{\mathtt{v}_1,\mathtt{t}_2'\}} \qquad \text{(E-Pair2)}$$

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

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

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

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

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-If)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \{\mathtt{t}_1,\mathtt{t}_2\} : \mathtt{T}_1 \times \mathtt{T}_2} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1\mathtt{.1} : \mathtt{T}_{11}} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \times \mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1\mathtt{.2} : \mathtt{T}_{12}} \qquad \text{(T-Proj2)}$$