# CIS 500 — Software Foundations

## Midterm II

## Answer key

**November 17, 2004**

## Simply typed lambda-calculus

*The following questions refer to the simply typed lambda-calculus with booleans and error. 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 the term does not type check, write NONE.

   (a) `λx:Bool→Bool→Bool. x true`

   Type: _____
   *Answer:* `(Bool→Bool→Bool) → Bool → Bool`

   (b) `λx:Bool. x x`

   Type: _____
   *Answer: NONE. There is no type that can be given to* `x` *to allow this term to type check.*

   (c) `λx:Bool→Bool. error x`

   Type: _____
   *Answer:* `(Bool→Bool) → Bool`

   (d) `λx:Bool→Bool. λy:Bool. x y error`

   Type: _____
   *Answer: NONE.* `x y` *has type* `Bool` *and may not be applied.*

   (e) `try (λx:Bool. if x then error else true) with false`

   Type: _____
   *Answer: NONE. Both parts of a* `try` *term must have the same type.*

   *Grading scheme: Binary. Two points per answer.*

2. (10 points) Which of the following functions *could* evaluate to error when applied to a single value of the appropriate type? Circle YES if there is some argument that could produce error and NO otherwise.

   (a) `λx:Bool. if x then true else false`

   YES NO

   *Answer: NO, both* `true` *and* `false` *arguments evaluate without error.*

   (b) `λx:Bool. if x then true else error`

   YES NO

   *Answer: YES, when applied to* `false` *the result is* `error`*.*

   (c) `λx:Bool. λy:Bool. if x then y else error`

   YES NO

   *Answer: NO, when applied to a* single *argument, the result is a lambda term.*

   (d) `λx:Bool→Bool. if (x true) then true else false`

   YES NO

   *Answer: YES, when applied to* `λy:Bool. error`*.*

   (e) `λx:Bool→Bool. if true then true else (x false)`

   YES NO

   *Answer: NO, returns* `true` *when applied to any value.*

   *Grading scheme: Binary. Two points per answer.*

3. (10 points) Suppose this language (the simply-typed lambda calculus with booleans and error, from page 1 of the companion handout) were extended with tuples, using the following syntax, evaluation and typing rules:

$$\begin{array}{rll} \mathtt{t} ::= & \ldots & \textit{terms:} \\ & \{\mathtt{t}_i \ ^{i \in 1..n}\} & \textit{tuple} \\ & \mathtt{t.i} & \textit{projection} \\ \\ \mathtt{v} ::= & \ldots & \textit{values:} \\ & \{\mathtt{v}_i \ ^{i \in 1..n}\} & \textit{tuple value} \\ \\ \mathtt{T} ::= & \ldots & \textit{types:} \\ & \{\mathtt{T}_i \ ^{i \in 1..n}\} & \textit{tuple type} \end{array}$$

$$\{\mathtt{v}_i \ ^{i \in 1..n}\}.\mathtt{j} \longrightarrow \mathtt{v}_j \qquad \text{(E-ProjTuple)}$$

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

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}_j'}{\{\mathtt{v}_i \ ^{i \in 1..j-1}, \mathtt{t}_j, \mathtt{t}_k \ ^{k \in j+1..n}\} \longrightarrow \{\mathtt{v}_i \ ^{i \in 1..j-1}, \mathtt{t}_j', \mathtt{t}_k \ ^{k \in j+1..n}\}} \qquad \text{(E-Tuple)}$$

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

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

(a) What additional rules must be added to the operational semantics so that the Progress Theorem is true? Recall that the Progress Theorem states: If $\emptyset \vdash \mathtt{t} : \mathtt{T}$ then either $\mathtt{t}$ is a value, $\mathtt{t}$ is `error`, or there exists a $\mathtt{t}'$ such that $\mathtt{t} \longrightarrow \mathtt{t}'$.

*Answer:*

$$\mathit{error.i} \longrightarrow \mathit{error} \qquad \text{(E-ProjError)}$$

$$\{\mathtt{v}_i \ ^{i \in 1..j-1}, \mathit{error}, \mathtt{t}_k \ ^{k \in j+1..n}\} \longrightarrow \mathit{error} \qquad \text{(E-TupleError)}$$

*Grading scheme: 2 points per rule. Partial credit given for small errors such as*

$$\frac{\mathtt{t}_1 \longrightarrow \mathit{error}}{\mathtt{t}_1.\mathtt{i} \longrightarrow \mathit{error}}$$

(b) Complete the inversion lemma for tuples and projections in this calculus.
If $\Gamma \vdash \{t_i\ ^{i \in 1..n}\} : R$ then ...
*Answer:* $R = \{T_i\ ^{i \in 1..n}\}$ *and for each* $i$, $\Gamma \vdash t_i : T_i$.

If $\Gamma \vdash t.j : R$ then ...
*Answer:* $\Gamma \vdash t : \{T_i\ ^{i \in 1..n}\}$ *and* $R = T_j$ *for some* $1 \leq j \leq n$.

*Grading scheme: 2 points each.*

(c) Complete the canonical forms lemma for tuples in this calculus.
If $v$ is a closed value of type $\{T_i\ ^{i \in 1..n}\}$ then ...
*Answer:* $v$ *is a tuple of the form* $\{v_i\ ^{i \in 1..n}\}$

*Grading scheme: 1 point for general form of a canonical forms lemma. 1 point for stating the correct lemma exactly. For example,* $v$ *is a tuple of the form* $\{t_i\ ^{i \in 1..n}\}$ *was worth 1 point.*

4. (9 points) In the simply-typed lambda calculus with error, booleans, let and tuples, we can encode binary sums as a derived form.

```
inl t₁ as T₁+T₂ ≝ {true, (λx:T₁. λa:Bool. x)t₁,   λa:Bool. error}
inr t₂ as T₁+T₂ ≝ {false, λa:Bool. error,   (λx:T₂. λa:Bool. x)t₂}
case t of inl x₁ ⇒ t₁ | inr x₂ ⇒ t₂ ≝
   let y = t in
   if y.1 then [x₁ ↦ (y.2 true)] t₁
          else [x₂ ↦ (y.3 true)] t₂
```

The following similar encodings are *incorrect*. Concisely describe why, using one or two sentences.

(a)
```
inl t₁ as T₁+T₂ ≝ {true, t₁, error}
inr t₂ as T₁+T₂ ≝ {false, error, t₂}
case t of inl x₁ ⇒ t₁ | inr x₂ ⇒ t₂ ≝
   let y = t in
   if y.1 then [x₁ ↦ y.2] t₁
          else [x₂ ↦ y.3] t₂
```

*Answer: This encoding immediately evaluates to* `error`*, so it is impossible to determine whether the first component of the tuple is true or false.*

(b)
```
inl t₁ as T₁+T₂ ≝ {true, t₁, (λa:Bool. error)}
inr t₂ as T₁+T₂ ≝ {false, (λa:Bool. error), t₂}
case t of inl x₁ ⇒ t₁ | inr x₂ ⇒ t₂ ≝
   let y = t in
   if y.1 then [x₁ ↦ y.2] t₁
          else [x₂ ↦ y.3] t₂
```

*Answer: The encodings of* `inl` *and* `inr` *don't produce terms of the same type.*

(c)
```
inl t₁ as T₁+T₂ ≝ {true, (λa:Bool. t₁), (λa:Bool. error)}
inl t₂ as T₁+T₂ ≝ {false, (λa:Bool. error), (λa:Bool.t₂)}
case t of inl x₁ ⇒ t₁ | inr x₂ ⇒ t₂ ≝
   let y = t in
   if y.1 then [x₁ ↦ (y.2 true)] t₁
          else [x₂ ↦ (y.3 true)] t₂
```

*Answer: The expression* $t_1$ *is only evaluated when the sum is examined, not when it is created. For example* `inl error as Bool + Bool` *should produce an error, but would not with this encoding.*

*Grading scheme: 3 points each.*

## References

*The following questions refer to the simply typed lambda-calculus with references (and* `Unit`, `Nat`, `Bool`, *and* `let`*). The syntax, typing, and evaluation rules for this system are given on page 3 of the companion handout.*

5. (12 points) Evaluating the expression

```
let x = ref (λn:Nat. 0) in
let y = ref (λn:Nat. (!x) n) in
let z = ref (λn:Nat. (!y) n) in
(!z) 3
```

beginning in an empty store yields:

Result: 0 Store: $l_1 \mapsto$ `λn:Nat. 0`
$l_2 \mapsto$ `λn:Nat. (!`$l_1$`) n`
$l_3 \mapsto$ `λn:Nat. (!`$l_2$`) n`

Fill in the resulting values and final stores (when started with an empty store) for the following terms:

(a)
```
let x = ref 0 in
let y = ref x in
(!y) := 3;
x
```

*Answer:*

*Result:* $l_1$ *Store:* $l_1 \mapsto$ `3`
$l_2 \mapsto l_1$

(b)
```
let x = ref 0 in
let f = ref (λw:Ref Nat. w:= succ (!x)) in
let g = λa:Nat. (x := succ(a); ref a) in
   (!f) (g (!x))
```

*Answer:*

*Result:* `unit` *Store:* $l_1 \mapsto$ `1`
$l_2 \mapsto$ `λw:Ref Nat. w:= succ (!`$l_1$`)`
$l_3 \mapsto$ `2`

(c)
```
let f = ref (λn:Nat. ref 500) in
f := λn:Nat. if iszero(n) then ref 0
                          else ref ( succ (!((!f) (pred n)) ));
(!f) 3
```

*Answer:*

*Result:* $l_5$ *Store:* $l_1 \mapsto$ `λn:Nat. if iszero(n) then (ref 0)`
`else ref (succ (!((!`$l_1$`)(pred n))))`
$l_2 \mapsto$ `0`
$l_3 \mapsto$ `1`
$l_4 \mapsto$ `2`
$l_5 \mapsto$ `3`

*Grading scheme: One point for the result, three points for the store.*

6. (8 points) Recall that the preservation theorem for the simply-typed lambda-calculus with references is stated as follows:

   **THEOREM** [Preservation]: If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mu$ and $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$ then, for some $\Sigma' \supseteq \Sigma$, $\Gamma \mid \Sigma' \vdash \mathtt{t}' : \mathtt{T}$ and $\Gamma \mid \Sigma' \vdash \mu'$.

   This theorem must be stated very carefully. Small changes to it can easily make it incorrect. For each of the FALSE variants of the statement shown below, write down a counter-example that demonstrates why that statement is incorrect.

   For example, the statement:

   If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mu$ and $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$ then $\Gamma \mid \Sigma \vdash \mathtt{t}' : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mu'$.

   is false because although $\emptyset \mid \emptyset \vdash \mathtt{ref}\ 0 : \mathtt{Ref\ Nat}$ and $\emptyset \mid \emptyset \vdash \emptyset$ and $\mathtt{ref}\ 0 \mid \emptyset \longrightarrow \mathtt{l} \mid \mathtt{l} \mapsto 0$,

   it is not the case that $\emptyset \mid \emptyset \vdash \mathtt{l} : \mathtt{Ref\ Nat}$.

   (a) If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$ then, for some $\Sigma' \supseteq \Sigma$, $\Gamma \mid \Sigma' \vdash \mathtt{t}' : \mathtt{T}$ and $\Gamma \mid \Sigma' \vdash \mu'$.
   *Answer:* $\emptyset \mid \mathtt{l} : \mathtt{Nat} \vdash !\mathtt{l} : \mathtt{Nat}$, *and* $!\mathtt{l} \mid \mathtt{l} = \mathtt{true} \longrightarrow \mathtt{true} \mid \mathtt{l} = \mathtt{true}$.
   *However, there is no* $\Sigma' \supseteq \mathtt{l} : \mathtt{Nat}$ *such that* $\emptyset \mid \Sigma' \vdash \mathtt{true} : \mathtt{Nat}$

   (b) If $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \mid \Sigma \vdash \mu$ and $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$ then, for all $\Sigma' \supseteq \Sigma$, $\Gamma \mid \Sigma' \vdash \mathtt{t}' : \mathtt{T}$ and $\Gamma \mid \Sigma' \vdash \mu'$.
   *Answer:* $\emptyset \mid \emptyset \vdash \mathtt{ref}\ 0 : \mathtt{Ref\ Nat}$, *and* $\emptyset \mid \emptyset \vdash \emptyset$ *and* $\mathtt{ref}\ 0 \mid \emptyset \longrightarrow \mathtt{l} \mid \mathtt{l} = \mathtt{Ref\ Nat}$.
   *However, it is not the case that an arbitrary extension* $\Sigma$ *of the empty context* $\emptyset$ *will allow the derivation of* $\emptyset \mid \Sigma \vdash \mathtt{l} : \mathtt{Ref\ Nat}$, *only those extensions that map* $\mathtt{l}$ *to* $\mathtt{Nat}$.

   *Grading scheme: Each part worth four points. 0 pts for a blank answer, 1 point if realized the statement was false, 2 points if realized a counterexample was needed, 3 points if the counterexample was almost correct and 4 points for producing a valid counterexample.*

7. (21 points) Intuitively, type soundness for a language states that if a closed term is well-typed and evaluates (using multi-step evaluation) to a normal form, that normal form will be a value. In other words, closed well-typed terms will not get stuck. We can state this idea precisely for the language with references in the following manner:

   **THEOREM** [Type Soundness]: If $\emptyset \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\emptyset \mid \Sigma \vdash \mu$ and $\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t}' \mid \mu'$ and $\mathtt{t}' \mid \mu' \not\longrightarrow$, then $\mathtt{t}'$ is a value.

   Recall also to the progress theorem for the simply-typed lambda calculus with references:

   **THEOREM** [Progress]: If $\emptyset \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ and $\emptyset \mid \Sigma \vdash \mu$ then either $\mathtt{t}$ is a value, or else, for any $\mu$ such that $\emptyset \mid \Sigma \vdash \mu$, there is some term $\mathtt{t}'$ and store $\mu'$ such that $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$.

   (a) Given the following definition of multi-step evaluation $\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t}' \mid \mu'$, prove type soundness by induction on this derivation using the progress and preservation theorems. Note, your proof must be concise as well as correct. Extraneous information not necessary for the proof (whether true or false) will be counted against you.

$$\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t} \mid \mu \qquad \text{(EV-DONE)}$$

$$\frac{\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu' \qquad \mathtt{t}' \mid \mu' \longrightarrow^* \mathtt{t}'' \mid \mu''}{\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t}'' \mid \mu''} \qquad \text{(EV-STEP)}$$

   *Answer: Proof is by induction on the derivation of $\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t}' \mid \mu'$. In each case we are trying to show that $\mathtt{t}'$ is a value.*

   - *case [EV-Done] Suppose $\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t} \mid \mu$. In this case $\mathtt{t}' = \mathtt{t}$ and $\mu' = \mu$. By the progress theorem, either $\mathtt{t}$ is a value or there is some $\mathtt{t}'$ and $\mu'$ such that $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'$. However, we know that $\mathtt{t} \mid \mu \not\longrightarrow$, so $\mathtt{t}$ must be a value.*
   - *case [EV-Step] Suppose that $\mathtt{t} \mid \mu \longrightarrow^* \mathtt{t}'' \mid \mu''$, where $\mathtt{t} \mid \mu \longrightarrow \mathtt{t}_1 \mid \mu_1$ and $\mathtt{t}_1 \mid \mu_1 \longrightarrow^* \mathtt{t}'' \mid \mu''$. In this case, $\mathtt{t}'=\mathtt{t}''$ and $\mu' = \mu''$.*
     *By the preservation theorem, we know that for some $\Sigma' \supseteq \Sigma$, $\emptyset \mid \Sigma' \vdash \mathtt{t}_1 : \mathtt{T}$ and $\emptyset \mid \Sigma' \vdash \mu_1$.*
     *Because we know that $\mathtt{t}_1$ and $\mu_1$ are well typed, we may apply induction and conclude that $\mathtt{t}''$ is a value.*

   *Grading scheme: The proof was worth 12 points. Roughly, 3 of those points were for setting up the cases correctly, 3 points for the EV-DONE case and 6 points for the EV-STEP case. Points were deducted for using the wrong lemma in the wrong case, for using the induction hypothesis incorrectly, for extraneous information and for minor mistakes or ommissions. No points were awarded for answers that tried to prove the wrong theorem (such as preservation or progress) and no points were awarded for students who tried to prove the lemma by induction on the the typing relation or single-step evaluation relation. (Although it may be possible to develop a convoluted proof in this manner, students who tried this route quickly lost track of what they were trying to prove.)*

(b) Suppose we eliminated the typing rule T-LOC from the type system of the simply typed lambda-calculus with references.

i. Is the preservation theorem still true? If so, give a *short* explanation why. If not, write down a counter-example.
*Answer: No, because the term* `ref 0` *is well-typed, but it steps to a location* `l` *that cannot be typed without T-LOC.*

ii. Is the progress theorem still true? If so, give a *short* explanation why. If not, write down a counter-example.
*Answer: Yes. Eliminating typing rules does not change the progress theorem. Although fewer closed terms type check, those that do (and are not values) must still step by the progress theorem of the original language.*

iii. Does type soundness still hold? If so, give a *short* explanation why. If not, write down a counter-example.
*Answer: Yes. The above proof is not the only way to prove type soundness. In this case, any term that type checks in the system without T-LOC also type checks in the original system. Type soundness of the original system specifies that the term must evaluate to a value, which is all that is required for type soundness to hold for the language without T-LOC. (Note that the value may not be typable without T-LOC, but that is not required by the type soundness theorem.)*

*Grading scheme: 1 point for the answer and 2 points for the counterexample or reason. Partial credit awarded for part iii if part i was wrong and the reason read that "since progress and preservation are still true, type soundness is still true."*

# Companion handout

# Full definitions of the systems used in the exam

## Simply typed lambda calculus with error (and `Bool`)

*Syntax*

| | | | |
|---|---|---|---|
| $t$ | ::= | | *terms* |
| | | $\texttt{error}$ | *run-time error* |
| | | $\texttt{true}$ | *constant true* |
| | | $\texttt{false}$ | *constant false* |
| | | $\texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t$ | *conditional* |
| | | $x$ | *variable* |
| | | $\lambda x{:}T.t$ | *abstraction* |
| | | $t\ t$ | *application* |
| $v$ | ::= | | *values* |
| | | $\texttt{true}$ | *true value* |
| | | $\texttt{false}$ | *false value* |
| | | $\lambda x{:}T.t$ | *abstraction value* |
| $T$ | ::= | | *types* |
| | | $T \to T$ | *type of functions* |
| | | $\texttt{Bool}$ | *type of booleans* |
| $\Gamma$ | ::= | | *contexts* |
| | | $\emptyset$ | *empty context* |
| | | $\Gamma, x{:}T$ | *term variable binding* |

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

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

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

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

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

$$\texttt{error } t_2 \longrightarrow \texttt{error} \qquad \text{(E-AppErr1)}$$

$$v_1 \texttt{ error} \longrightarrow \texttt{error} \qquad \text{(E-AppErr2)}$$

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash t_1 : \mathtt{Bool} \qquad \Gamma \vdash t_2 : T \qquad \Gamma \vdash t_3 : T}{\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : T} \quad \text{(T-If)}$$

$$\Gamma \vdash \mathtt{error} : T \quad \text{(T-Error)}$$

## Simply typed lambda calculus with references (and `Unit`, `Nat`, `Bool`, and `let`)

*Syntax*

| | | |
|---|---|---|
| t | ::= | *terms* |
| | x | *variable* |
| | let x=t in t | *let binding* |
| | unit | *constant* unit |
| | 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→T | *type of functions* |
| | Ref T | *type of reference cells* |
| | Bool | *type of booleans* |
| | Nat | *type of natural numbers* |
| μ | ::= | *stores* |
| | ∅ | *empty store* |
| | μ, l = v | *location binding* |
| Γ | ::= | *contexts* |
| | ∅ | *empty context* |
| | Γ, x:T | *term variable binding* |
| Σ | ::= | *store typings* |
| | ∅ | *empty store typing* |
| | Σ, l:T | *location typing* |

$$\begin{array}{lll} \mathsf{nv} ::= & & \textit{numeric values} \\ & \mathtt{0} & \textit{zero value} \\ & \mathtt{succ}\ \mathsf{nv} & \textit{successor value} \end{array}$$

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

$$\mathtt{let}\ x{=}v_1\ \mathtt{in}\ t_2 \mid \mu \longrightarrow [x \mapsto v_1]t_2 \mid \mu \qquad \text{(E-LetV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{let}\ x{=}t_1\ \mathtt{in}\ t_2 \mid \mu \longrightarrow \mathtt{let}\ x{=}t_1'\ \mathtt{in}\ t_2 \mid \mu'} \qquad \text{(E-Let)}$$

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

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

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

$$\frac{l \notin dom(\mu)}{\mathtt{ref}\ v_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto v_1)} \qquad \text{(E-RefV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{ref}\ t_1 \mid \mu \longrightarrow \mathtt{ref}\ t_1' \mid \mu'} \qquad \text{(E-Ref)}$$

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

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \qquad \text{(E-Deref)}$$

$$l{:=}v_2 \mid \mu \longrightarrow \mathtt{unit} \mid [l \mapsto v_2]\mu \qquad \text{(E-Assign)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1{:=}t_2 \mid \mu \longrightarrow t_1'{:=}t_2 \mid \mu'} \qquad \text{(E-Assign1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1{:=}t_2 \mid \mu \longrightarrow v_1{:=}t_2' \mid \mu'} \qquad \text{(E-Assign2)}$$

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

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

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

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{succ}\ t_1 \mid \mu \longrightarrow \mathtt{succ}\ t_1' \mid \mu'} \qquad \text{(E-Succ)}$$

$$\mathtt{pred}\ \mathtt{0} \mid \mu \longrightarrow \mathtt{0} \mid \mu \qquad \text{(E-PredZero)}$$

$$\mathtt{pred}\ (\mathtt{succ}\ nv_1) \mid \mu \longrightarrow nv_1 \mid \mu \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{pred}\ t_1 \mid \mu \longrightarrow \mathtt{pred}\ t_1' \mid \mu} \qquad \text{(E-Pred)}$$

$$\texttt{iszero}\ \texttt{0}|\ \mu \longrightarrow \texttt{true}|\ \mu \qquad \text{(E-IszeroZero)}$$

$$\texttt{iszero}\ (\texttt{succ}\ \mathsf{nv}_1)|\ \mu \longrightarrow \texttt{false}|\ \mu \qquad \text{(E-IszeroSucc)}$$

$$\frac{\mathtt{t}_1|\ \mu \longrightarrow \mathtt{t}_1'|\ \mu'}{\texttt{iszero}\ \mathtt{t}_1|\ \mu \longrightarrow \texttt{iszero}\ \mathtt{t}_1'|\ \mu'} \qquad \text{(E-IsZero)}$$

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

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

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

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

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

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

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

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

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

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

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

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

$$\Gamma \vdash \texttt{0} : \texttt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{succ}\ \mathtt{t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{pred}\ \mathtt{t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \texttt{let}\ \mathtt{x}{=}\mathtt{t}_1\ \texttt{in}\ \mathtt{t}_2 : \mathtt{T}_2} \qquad \text{(T-Let)}$$